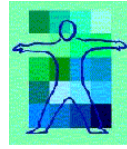


SIMILAR PhD Registration Number:

ISBN 978-2-87463-114-6

Copyright registration D/2008/9964/8

EAN 9782874631146



A Methodology for Developing Multimodal User Interfaces of Information Systems

By Adrian Stanciulescu

A dissertation submitted in fulfillment of the requirements
for the degree of

Doctor of Philosophy in
Management Sciences

of the Université catholique de Louvain

Committee in charge:

Prof. Jean Vanderdonckt, Université catholique Louvain, Advisor

Prof. Benoît Macq, Université catholique Louvain, Examiner

Prof. Manuel Kolp, Université catholique Louvain, Examiner

Prof. Philippe Palanque, Université Paul Sabatier, Reader

Prof. Yacine Bellik, Université Paris XI, Reader

Prof. Alain Vas, Université catholique Louvain, President of Jury

25 June 2008

Acknowledgements

I would like to express my thanks to:

- My advisor, Professor Jean Vanderdonckt, for his constant support and enthusiasm regarding my work. Collaborating with him was not only a permanently enriching professional experience, but also an opportunity to work with a person whose human side I highly appreciate.
- Professors Benoît Macq (Université catholique Louvain, TELE Laboratory, Belgium), Manuel Kolp (Université catholique Louvain, ISYS unit, Belgium), Philippe Palanque (LIIHS-IRIT, Université Paul Sabatier, France) and Yacine Bellik (LIMSI-CNRS, Université Paris XI, France) for accepting to participate to the jury of this thesis.
- My colleagues from the Louvain School of Management (LSM) of Université catholique de Louvain (UCL) and especially the ones sharing the same office who endured my long, noisy and most of the time annoying vocal tests.
- Familiei mele și în special părinților mei care au fost cu adevărat alături de mine în momentele de bucurie, dar mai ales în situațiile dificile pe parcursul studiilor universitare. Un sincer mulțumesc pentru felul în care ați știut să îmi îndrumați pașii prin sfaturile și experiența voastră de viață, lăsându-mi în același timp libertatea de a-mi lua propriile decizii.
- Prietenei mele Milena, pentru susținerea și încurajările continue care mi-au adus un nou suflu și mi-au permis să îmi regăsesc entuziasmul de-a lungul ultimilor ani de teză. Mulțumesc pentru optimismul de care dai dovadă și pe care ai știut să mi-l insufli atunci când aveam mai mare nevoie. Atitudinea ta mereu pozitivă m-a ajutat să depășesc mai ușor momentele dificile și să-mi clarific deciziile viitoare. Mulțumesc de asemenea pentru ajutorul acordat în redactarea tezei.

This thesis was accomplished thanks to the support of:

- The ISYS research unit of Louvain School of Management.

- The SIMILAR network of excellence, the European research task force creating human-machine interfaces similar to human-human communication, supported by the 6th Framework Program of the European Commission, under contract FP6-IST1-2003-507609 (www.similar.cc).

- The OpenInterface Foundation, supported by the 6th Framework Program of the European Commission, under the umbrella of SIMILAR network of excellence (www.openinterface.com).

- The UsiXML Consortium, for User Interface eXtensible Markup Language (www.usixml.org).

- Philippe Guillaume, Didier Magotteaux and Vincent Wauters (IBM Belgium) for the grant received for the IBM Multimodal Toolkit™, Software Modeler™, and WebSphere™, with which this research has been achieved.

Table of Contents

ACKNOWLEDGEMENTS	I
TABLE OF CONTENTS	1
TABLE OF FIGURES	6
TABLE OF TABLES	10
1 INTRODUCTION	13
1.1 Context.....	13
1.2 Concerns of multimodal user interfaces.....	14
1.3 Terminology used in this thesis	16
1.3.1 Mode.....	17
1.3.2 Media.....	17
1.3.3 Modality	17
1.3.4 Multimode, multimedia and multimodality.....	18
1.4 Thesis	18
1.4.1 Thesis statement	18
1.4.2 Definitions of working hypotheses.....	19
1.4.3 Scope	22
1.5 Reading Map.....	24
2 STATE OF THE ART	27
2.1 Introduction	27
2.2 A structuring theoretical framework.....	27
2.2.1 TYCOON framework.....	28
2.2.2 CARE properties.....	29
2.2.3 W3C Multimodal Interaction Framework	31
2.2.4 Comparison of theoretical frameworks	33
2.3 User Interface Description Languages	33
2.3.1 XISL	33
2.3.2 XIML	34
2.3.3 UIML	35
2.3.4 DISL	36

2.3.5	VoiceXML.....	36
2.3.6	XHTML+Voice	37
2.3.7	TeresaXML.....	38
2.3.8	EMMA.....	38
2.4	User interface development tools	38
2.4.1	Galatea Interaction Builder.....	38
2.4.2	UIML Development Toolkit.....	39
2.4.3	WebSphere Voice Toolkit	40
2.4.4	Teresa	40
2.4.5	MONA	41
2.4.6	SUEDE	42
2.4.7	CSLU Toolkit	43
2.4.8	MOST	43
2.4.9	OpenInterface Platform	44
2.4.10	A Toolkit of Multimodal Widgets.....	45
2.4.11	FRUIT.....	48
2.5	Conclusion	50
2.5.1	Summary of the state of the art.....	50
2.5.2	Shortcomings	54
2.5.3	Requirements	55
2.6	Conclusion	57
3	CONCEPTUAL MODELING OF MULTIMODAL USER INTERFACES.....	59
3.1	Introduction	59
3.2	Selection of a User Interface Description Language	59
3.2.1	Towards choosing a suitable UIDL	59
3.2.2	UsiXML – the selected UIDL	60
3.3	Conceptual contribution	61
3.3.1	Task Model.....	62
3.3.2	Domain Model.....	65
3.3.3	Abstract User Interface Model	66
3.3.4	Concrete User Interface Model.....	71
3.3.5	Mapping Model	72
3.3.6	Transformation Model.....	72
3.4	Semantics of the multimodal interaction objects.....	74
3.4.1	Semantics of the Graphical Concrete Interaction Objects.....	74
3.4.2	Semantics of the Vocal Concrete Interaction Objects	76
3.4.3	Semantics of the Multimodal Concrete Interaction Objects.....	84
3.4.4	Semantics of the Concrete User Interface Relationships.....	89
3.5	Syntax of UsiXML	91

3.5.1	From Semantics to Concrete Syntax	91
3.5.2	Concrete Syntax of Interaction Objects	93
3.6	Stylistics of interaction objects	96
3.7	Conclusions	98
4	A TRANSFORMATIONAL METHOD FOR PRODUCING MULTIMODAL USER INTERFACES.....	99
4.1	Introduction	99
4.2	Design space for user interfaces	99
4.2.1	Definition of design space	99
4.2.2	Rationale for choosing a design space.....	100
4.2.3	Design options for user interfaces	102
4.2.4	Design space in the context of Design Rationale approach.....	121
4.3	Specification of transformations	124
4.3.1	Selection of model-to-model transformational approach	124
4.3.2	Application strategy of transformation rules	130
4.3.3	Shortcomings of the existing graph-based transformational approaches	131
4.3.4	Expanded model-to-model transformational approach	131
4.3.5	Transformation rule catalog.....	138
4.4	The four steps of the transformational approach.....	141
4.4.1	Step 1: The Task and Domain Models	143
4.4.2	Step 2: From Task and Domain Models to Abstract User Interface Model	143
4.4.3	Step 3: From Abstract User Interface Model to Concrete User Interface Model	147
4.4.4	Step 4: From Concrete User Interface Model to Final User Interface.....	154
4.5	Conclusion	154
5	TOOL SUPPORT	155
5.1	Introduction	155
5.2	MultimodalXML modules	156
5.2.1	IdealXML	156
5.2.2	TransformiXML	157
5.2.3	GrafiXML	159
5.2.4	VoiceXML Generator	160
5.2.5	XHTML+Voice Generator	160
5.3	Limitations of current tool support	161
5.4	Conclusions	163

6	VALIDATION	165
6.1	Introduction	165
6.2	Case study 1: Virtual Polling Application.....	165
6.2.1	Step 1: The Task and Domain Models	166
6.2.2	Step 2: From Task and Domain Models to AUI Model	170
6.2.3	Step 3: From AUI Model to CUI Model	172
6.2.4	Step 4: From CUI Model to FUI	177
6.3	Case study 2: Car Rental Application	179
6.3.1	Step 1: The Task and Domain Models	180
6.3.2	Step 2: From Task and Domain Models to AUI Model	184
6.3.3	Step 3: From AUI Model to CUI Model	185
6.3.4	Step 4: From CUI Model to FUI	193
6.4	Case study 3: Map Browsing Application	195
6.4.1	Step 1: The Task and Domain Models	196
6.4.2	Step 2: From Task and Domain Models to AUI Model	198
6.4.3	Step 3: From AUI Model to CUI Model	199
6.4.4	Step 4: From CUI Model to FUI	204
6.5	Empirical validation.....	206
6.5.1	Methodology usability assessment	206
6.5.2	Methodology result assesment plan.....	212
6.5.3	Results	214
6.5.4	Interpretation and discussion	221
6.6	Internal validation	227
6.7	Conclusion	231
6.7.1	Conclusions issued from the external validation	231
6.7.2	Conclusions issued from the internal validation.....	234
7	CONCLUSION	235
7.1	Introduction	235
7.2	Summary of contributions	235
7.2.1	Theoretical and conceptual contributions.....	235
7.2.2	Methodological contribution	236
7.2.3	Tools developed.....	239
7.3	Future work in prospect.....	239
7.4	Some personal concerns	241
7.5	Concluding remarks	243

REFERENCES	245
APPENDIX A. USIXML EXPANDED TASK MODEL	255
APPENDIX B. TRANSFORMATION RULE CATALOG	257
APPENDIX C. USIXML CONCRETE SYNTAX FOR THE SPECIFICATION OF DIFFERENT COMBINATIONS OF INPUT AND OUTPUT MODALITIES.....	295
APPENDIX D. QOC REPRESENTATION OF DESIGN SPACE OPTIONS IN TEAM TOOL.....	307
APPENDIX E. ACRONYMS	313

Table of Figures

Figure 1-1 “Put that there” multimodal system	14
Figure 1-2 Benefits of the identified concerns of multimodal UIs	16
Figure 1-3 The human five senses	17
Figure 1-4 Capabilities vs. resources for producing a user interface	21
Figure 1-5 Thesis structure	24
Figure 2-1 General schema for state of the art analysis	27
Figure 2-2 W3C Multimodal Interaction Framework	32
Figure 2-3 Multimodal X+V application interpreted with Opera browser	37
Figure 2-4 The Interaction Builder graphical user interface	39
Figure 2-5 UIML Development Tool	39
Figure 2-6 IBM WebSphere Voice Toolkit – communication flow builder perspective ...	40
Figure 2-7 Authoring a multimodal UI with Teresa	41
Figure 2-8 MONA editor with real time GUI previews	41
Figure 2-9 Design mode in SUEDE	42
Figure 2-10 CSLU toolkit - the graphical authoring editor	43
Figure 2-11 Interaction component editor in MOST	44
Figure 2-12 Integration of heterogenous components in OpenInterface	45
Figure 2-13 Multimodal toolkit architecture	47
Figure 2-14 The toolkit architecture –button feedback to mouse-over event	48
Figure 2-15 The architecture of a FRUIT system	50
Figure 3-1 Cameleon Reference Framework for multi-target UIs	60
Figure 3-2 Meta-model of the Task Model	63
Figure 3-3 Meta-model of the Domain Model	66
Figure 3-4 Meta-model of the AUI Model	67
Figure 3-5 Abstract attribute values inheriting Task attribute values	68
Figure 3-6 The general structure of an instruction in ISs	68
Figure 3-7 Excerpt of the CUI Meta-model	71
Figure 3-8 Meta-model of the Mapping Model	72
Figure 3-9 Meta-model of the Transformation Model	74
Figure 3-10 Graphical containers	75
Figure 3-11 Several Graphical Individual Components	76
Figure 3-12 Vocal Concrete Interaction Objects	82
Figure 3-13 VocalCIOs involved in the fulfillment of Provide age task	83
Figure 3-14 VocalCIOs used for a vocal application of a Phone line company	84
Figure 3-15 Concrete UI Relationships	91
Figure 3-16 Generation of UsiXML specification	92
Figure 3-17 Transforming of a class to into UsiXML specification	92
Figure 3-18 Transformation of a relationship class into UsiXML specification	92
Figure 3-19 Transformation of the inheritance relationship into UsiXML specification ..	93
Figure 3-20 Transformation of the aggregation relationship into UsiXML specification ..	93
Figure 4-1 The design options composing the design space	102

Figure 4-2 Sub-task presentation values.....	105
Figure 4-3 Sub-task presentation values and a possible concretization in vocal and graphical objects.....	107
Figure 4-4 Types of navigation between sub-tasks and a possible concretization in vocal and graphical objects.....	109
Figure 4-5 Navigation type values and a possible concretization in vocal and graphical objects.....	110
Figure 4-6 Control type values and a possible concretization in vocal and graphical objects.....	111
Figure 4-7 Navigation and control type values and a possible concretization in vocal and graphical objects.....	112
Figure 4-8 Guided sub-task in GUI	113
Figure 4-9 Unguided sub-task in GUI	113
Figure 4-10 Ambiguous answer in GUI	114
Figure 4-11 Unambiguous answer in GUI	114
Figure 4-12 Single answer in GUI	115
Figure 4-13 Multiple answer in GUI	115
Figure 4-14 Confirmation message in GUI	116
Figure 4-15 Non-confirmed message in GUI	116
Figure 4-16 Order dependent answer in GUI	117
Figure 4-17 Order independent answer in GUI	117
Figure 4-18 A possible design decision for a multimodal text input.....	121
Figure 4-19 QOC diagram structure.....	123
Figure 4-20 QOC representation of the sub-task guidance design option.....	123
Figure 4-21 QOC representation of the sub-task guidance design option.....	124
Figure 4-22 Progressive application of rule-based transformations.....	125
Figure 4-23 Identification of transformation rule approach features.....	126
Figure 4-24 Structure of a transformation catalog.....	126
Figure 4-25 Characterization of transformation in UsiXML.....	127
Figure 4-26 Syntactically typed patterns and variables.....	128
Figure 4-27 Graphical concrete syntax of the patterns.....	128
Figure 4-28 From Task Model to Abstract Model.....	129
Figure 4-29 Textual syntax for expressing transformation rules.....	129
Figure 4-30 The application strategy of transformation rules.....	130
Figure 4-31 Monocolored transformation rule generating: (a) groupBox elements; (b) vocalMenu elements; (c) groupBoxes and vocalMenu elements.....	137
Figure 4-32 Transformation rules supporting sub-task presentation.....	138
Figure 4-33 Transformation rules supporting the vocal and graphical concretization of sub-task presentation values.....	139
Figure 4-34 Transformation rules supporting sub-task navigation values for graphical and vocal concretization.....	139
Figure 4-35 Transformation rules supporting navigation type values for graphical and vocal concretization.....	140
Figure 4-36 Transformation rules supporting control type values for graphical and vocal concretization.....	140
Figure 4-37 Transformation rules supporting navigation and control type values for graphical and vocal concretization.....	141
Figure 4-38 Transformation rules supporting the remaining set of design options for which a stylistics was not assigned.....	141
Figure 4-39 General development scenario of UI.....	142

Figure 4-40 Sub-steps of the transformational approach.....	143
Figure 4-41 Separated sub-task presentation	Figure 4-42 Combined sub-task presentation.....
.....	144
Figure 4-43 Local placement for navigation	Figure 4-44 Global placement for navigation
.....	145
Figure 4-45 Local placement for control	Figure 4-46 Global placement for control.....
.....	145
Figure 4-47 Sequential navigation between sub-tasks presented in separated windows..	150
Figure 4-48 Asynchronous navigation between sub-tasks presented in separated windows	150
Figure 5-1 General development scenario – identification of MultimodalXML modules	155
Figure 5-2 Task Model editor	Figure 5-3 Domain Model editor
.....	156
Figure 5-4 Mapping Model editor	156
Figure 5-5 TransformiXML – graphical user interface	157
Figure 5-6 Model-to-model transformation based on AGG API.....	158
Figure 5-7 Generate abstract containers for each sub-task of the top-most task	158
Figure 5-8 Initial Model	Figure 5-9 Resultant model.....
.....	158
Figure 5-10 GrafiXML – export function.....	159
Figure 5-11 Multimodal design tools complexity vs. specificity	162
Figure 5-12 A design space-based tool for development of multimodal UIs.....	163
Figure 6-1 Development scenario for virtual polling application	166
Figure 6-2 Mappings between the Task Model and the Domain Model	166
Figure 6-3 Mapping Model for the virtual polling system	167
Figure 6-4 Task Model expressed in UsiXML	168
Figure 6-5 Domain Model expressed in UsiXML.....	169
Figure 6-6 Mapping Model expressed in UsiXML	169
Figure 6-7 AUI Model expressed in UsiXML.....	171
Figure 6-8 Graphical UI	Figure 6-9 Vocal UI.....
.....	178
Figure 6-10 Multimodal UI	179
Figure 6-11 Development scenario for car rental application	180
Figure 6-12 The decomposition of Determine rental preferences sub-task.....	180
Figure 6-13 The decomposition of Determine rental preferences sub-task.....	180
Figure 6-14 The decomposition of Determine car sub-task	181
Figure 6-15 The decomposition of Provide payment information sub-task	181
Figure 6-16 Excerpt of Task Model expressed in UsiXML	182
Figure 6-17 Domain Model for the car rental system.....	183
Figure 6-18 Excerpts of Domain Model expressed in UsiXML.....	183
Figure 6-19 FUI – graphical input	Figure 6-20 FUI – vocal input.....
.....	194
Figure 6-21 FUI – equivalent graphical and vocal input.....	195
Figure 6-22 Development scenario for the map brosing system	196
Figure 6-23 Task Model of the map browsing application.....	196
Figure 6-24 Specification of the Task Model in UsiXML	197
Figure 6-25 Domain Model for the map browsing system.....	197
Figure 6-26 Excerpts of Domain Model expressed in UsiXML.....	198
Figure 6-27 FUI – graphical input.....	204

Figure 6-28 FUI – vocal input	205
Figure 6-29 FUI – graphical input for browsing action and vocal input for browsing direction	205
Figure 6-30 Methodology assessment levels.....	206
Figure 6-31 The multimodal version of the DVD rental application	210
Figure 6-32 Physical position of the subjects and experimental apparatus	211
Figure 6-33 A subject interacting with the application	212
Figure 6-34 The training application	213
Figure 6-35 Task completion mean time per interaction modality.....	215
Figure 6-36 Mean task procentage completion per interaction modality	215
Figure 6-37 Mean number of errors per category.....	217
Figure 6-38 The timeline for a correct voice-enabling button manipulation.....	217
Figure 6-39 Learning time for vocal interaction	218
Figure 6-40 Mean number of mouse clicks per interaction modality.....	218
Figure 6-41 Distribution of the modality preference per subject	219
Figure 6-42 Mean number of errors per experience group for vocal interaction	221
Figure 6-43 Mean number of errors per experience group for MM interaction	221
Figure 6-44 Distribution of task completion time per interaction type and experience group.....	222
Figure 6-45 Task completion mean time per experience group	223
Figure 6-46 Modality interaction preference per application type	225
Figure 6-47 Requirements coverage rate.....	234
Figure 7-1 Connection between the effort rate and the outcome rate of our methodology	242

Table of Tables

Table 2-1 Comparison of the surveyed user interface description languages	53
Table 3-1 Definition of existing values for the userAction attribute.....	62
Table 3-2 Definitions of existing values for the taskItem attribute	62
Table 3-3 Definition of newly identified values for the taskType attributes.....	64
Table 3-4 Synonyms for the taskType values	64
Table 3-5 Definitions of newly identified values for the taskItem attribute.....	64
Table 3-6 Examples of combinations between values of taskType and taskItem attributes	65
Table 3-7 Correspondence between popular widgets and CIOs of different modalities	88
Table 3-8 Possible combinations of input/output interaction types for a label and a textFiled	94
Table 3-9 Stylistics for several vocal concrete interaction objects.....	98
Table 4-1 Color associated to the UsiXML model concepts	133
Table 4-2 Mappings between tasks types and AIC facets types.....	146
Table 4-3 Mappings between facet types and GIC types	148
Table 4-4 Design option values for textInput widget with graphical assignement for input	149
Table 4-5 Mappings between facet types and VIC types	151
Table 4-6 Design option values for vocal assigned input.....	152
Table 4-7 Mappings between facet types and GIC and VIC types.....	153
Table 4-8 Design option values for multimodal textInput widget (graphical and vocal equivalence for input).....	153
Table 5-1 Mappings between the vocal CIOs and VoiceXML elements	160
Table 5-2 Mappings between the graphical CIOs and the XHTML elements	161
Table 6-1 Design option values for inputText.....	172
Table 6-2 Design option values for radioButtons.....	173
Table 6-3 Design option values for outputText.....	173
Table 6-4 Design option values for control buttons	173
Table 6-5 Design option values for vocalInput	174
Table 6-6 Design option values for vocalInput with confirmation.....	174
Table 6-7 Design option values for vocalInput with grammar items	175
Table 6-8 Design option values for vocalPrompt.....	175
Table 6-9 Design option values for submit element.....	175
Table 6-10 Design option values for multimodal inputText	176
Table 6-11 Design option values for multimodal radioButtons	177
Table 6-12 Design option values for outputText.....	177
Table 6-13 Mappings between task and domain models.....	184
Table 6-14 Design option values for inputText.....	186
Table 6-15 Design option values for comboBox.....	186
Table 6-16 Design option values for radioButtons.....	187
Table 6-17 Design option values for checkBoxes	187
Table 6-18 Design option values for listBox.....	187
Table 6-19 Design option values for multimodal inputText	188

Table 6-20 Design option values for multimodal combobox	189
Table 6-21 Design option values for multimodal radioButtons	189
Table 6-22 Design option values for multimodal checkBoxes.....	189
Table 6-23 Design option values for multimodal listBoxes	190
Table 6-24 Design option values for multimodal inputText	191
Table 6-25 Design option values for multimodal combobox	192
Table 6-26 Design option values for multimodal radioButtons	192
Table 6-27 Design option values for multimodal checkBoxes.....	192
Table 6-28 Design option values for multimodal listBoxes	193
Table 6-29 Mappings between task and domain models.....	198
Table 6-30 Design option values for multimodal radioButtons	200
Table 6-31 Design option values for imageZones.....	200
Table 6-32 Design option values for multimodal radioButtons	201
Table 6-33 Design option values for multimodal radioButtons	203
Table 6-34 Design option values for multimodal radioButtons	203
Table 6-35 Estimated learning time of the methodological aspects	207
Table 6-36 Summary of the subject's demographics and experience level.....	209
Table 6-37 Mean task completion time (seconds) per experience group	220
Table 6-38 Mean task procentage completion per experience group	220
Table 6-39 t-Test results for the significant difference in mean task completion time with respect to MM experience	222
Table 6-40 t-Test results for the significant difference in mean number of pronunciation and synchronization errors with respect to MM experience.....	223
Table 6-41 t-Test results for the significant difference in mean completion time with respect to experience category.....	224
Table 6-42 Subject's opinion for web form applications	226
Table 6-43 Subject's opinion for non web form application.....	227
Table 7-1 Dependencies between design options.....	237

1 Introduction

1.1 Context

The most prevailing type of User Interface (UI) in today's interactive applications is the Graphical User Interface (GUI). Since GUIs restrict the Human-Computer Interaction (HCI) mainly to the visual mode, they do not allow end users to communicate in ways they naturally do with other human beings [Klem00]. More particularly, the standard GUI does not work well for some users (e.g., users having limited literacy or typing skills), in some circumstances (e.g., when users are moving around, when their hands or their eyes are busy with other tasks), when the environment is constrained (e.g., the keyboard and the mouse are not available) or when the end user is interacting with another person. In order to go beyond the GUI imposed limitations, a new UI paradigm is needed. Multimodal (MM) UIs is one of these paradigms having the expected capabilities.

The aforementioned problems also arise on the Internet, where an ever increasing portion of the user population is carrying out interactive tasks with more advanced interaction devices (e.g., mobile phones, smart phones, Personal Digital Assistants – PDAs). As this population portion is increasing, new specific needs should be addressed. As interaction devices become smaller, means of input other than keyboard or tap screen become necessary. Indeed these devices benefit nowadays of enough processing power to handle multiple and complex tasks. This situation also leads to considering a new application technology called *multimodal*, where multiple methods of communication between the end user and interaction devices are considered simultaneously. These methods include, but are not limited to: keypad, tap screen, tactile screen, handwriting recognition, speech synthesis, voice recognition, and gesture recognition.

MM UIs represent a research-level paradigm shift away from conventional windows-icons-menus-pointers (WIMP) interfaces towards providing users with great expressive power, naturalness, flexibility and portability [Ovia99]. Such flexibility makes it possible for users to alternate modalities so that physical overexertion is avoided for any individual modality. It also permits substantial error avoidance and easier error recovery. The flexibility of a MM interface can accommodate a wide range of users, tasks and environments. For example, users who are temporarily or permanently disabled, tasks which were not possible to carry out before and environments in adverse or very constrained settings (e.g., noisy environments, mobile conditions) when a single mode may not suffice. In many of these real-world examples, integrated MM UIs exhibit the potential to support entirely new capabilities that have not been envisioned by previous traditional systems based on GUIs.

1. Introduction

MM UIs have been viewed as an attractive area for HCI research since Bolt's seminal "Put That There" system [Bolt80] where graphical objects are created and moved on a wall screen using speech recognition and finger pointing (Figure 1-1).



Figure 1-1 "Put that there" multimodal system

Since then, the promise of MM UIs to deliver a more natural and efficient interaction has not been discontinued [Cohe98]. MM UIs are expanding both in popularity due to the increasing accuracy of perceptual input systems (e.g., voice recognition, handwriting recognition, vision recognition) and the increasing ubiquity of heterogeneous computing platforms (e.g., mobile telephones, handheld devices, laptops, whiteboards) and in the range of information systems they support:

- Accessing business information, support desks, order tracking, airline arrival and departure information, cinema and theater booking services and home banking.
- Accessing public information, including community information such as weather, traffic conditions, school closures, directions and events; local, national and international news; national and international stock market information; and business and e-commerce transactions.
- Accessing personal information, including calendars, address and telephone lists, to-do lists, shopping lists and calorie counters.

Since more and more people have access to the Internet, MM UIs promise to enable anyone to access web based information systems from any online computing platform, mobile or stationary, from anywhere and at anytime (e.g., at work, at home, on the move between).

1.2 Concerns of multimodal user interfaces

In the context of this thesis we identify hereafter a set of concerns that are considered important for developing MM information systems:

- *Concern 1. Lack of support for multiple input/output modalities:* end users are not able to flexibly choose the most suitable interaction modality for their task, as its achievement depends on several aspects: the environment (e.g., noisy), the context of use (e.g., driving in a car), the task complexity (e.g., directory assistance), the

1. Introduction

- device capabilities (e.g., small displays), the users' disabilities (e.g., visual impairment) [Awde06].
- *Concern 2. Lack of separation of modalities:* most of the existing model-based approach do not provide a separation of concepts assigned to different modalities. This could enable designers to specify separately the UI corresponding to each modality and to further connect them altogether. Moreover, they could reuse, partially or totally, the specification corresponding to an interaction modality in other applications that employ it.
 - *Concern 3. Lack of combination of modalities:* the existing MM systems do not always provide a faster and more robust interaction as they rarely take advantage of the combination capabilities of interaction modalities characterizing such systems. For instance, they do not consider multiple modalities enabling parallel independent or complemetar input in order to achive the tasks. Moreover, the users are rarely able to select between two or more equivalent modalities the one they consider the fastest for the task to achieve.
 - *Concern 4. Lack of modality-independent model:* existing model-based approaches suffer from a lack of a modality-independent model in the development life cycle [Limb04b]. Due to the continously increasing number of new interaction devices and as a consequence of interaction modalities that will determine the development of new UIs with new modality capabilities, such model could enable to avoid their redeployment from scratch. In addition, it could contribute to the principle of separation of concerns [Dijk76].
 - *Concern 5. Lack of extensibility for new modalities:* nowadays, the constant emergence of new computing platforms supporting new sets of interaction modalities requires the intergration of new model concepts manipulated by methods. Currently, these concepts are difficult to extend therefore preventing the adaptation of methodologies for covering new interaction modalities.
 - *Concern 6. Lack of human readability of the ontology:* few methods define in an explicit manner their underlying concepts which are generally bounded to tools or methodological recommendations, thus preventing a designer to grasp the conceptual foundations of a methodology [Limb04b]. Moreover, research teams tend to conduct their researches and developments on their own models which make conceptual consolidation across methods difficult. Cross-method understanding is a tedious and time-consuming activity because it requires understanding the peculiarities of each method and establishing correspondence between them. As a consequence, communication among researchers becomes complex.
 - *Concern 7. Lack of a structuring framework for the development of MM UIs:* we are not aware of any development framework of MM UIs that structures the development life cycle in terms of options to select by designers. Currently, the designer's decisions are not explicitly defined and do not clarify the development of such systems which therefore requires more design workload.
 - *Concern 8. Lack of method explicitness:* existing approaches seriously lack explicitness in the way they propose their catalog of model-to-model and model-to-code transformations both to the designer and to researchers [Limb04b]. The transformation catalogs are often implicitly maintained in the head of developers and designers

1. Introduction

and/or hard-coded in supporting software. Consequently, the transformational processes proposed in the literature consist essentially of black boxes. This lack of explicitness dramatically hampers methodological guidance.

- *Concern 9. Lack of method extendibility:* developing UIs consists of making heuristic decisions in a vast design space. Transformations have consequently an inherent heuristic nature as they try to translate into algorithms part of these design decisions. Proposed methods offer very little possibilities to the designer to modify built-in heuristics: adding, deleting, modifying, reusing transformations is almost impossible [Limb04b].
- *Concern 10. Lack of support for tool interoperability:* consequently to the lack of explicitness, the exchange of knowledge regarding transformation catalogs can hardly be achieved [Limb04b]. Even when transformation catalogs are made explicit in tools, their heterogeneous formats prevents the reuse of transformations outside the context for which they were designed.

Under the light shed by the above set of concerns we benefit from a twofold result (Figure 1-2): (1) the statement of the current thesis is defined in Section 1.4.1, (2) a set of features of MM UIs are employed in Section 2.5.1 in order to analyse the user interface description languages (UIDLs) surveyed in the state of the art (Section 2.3).

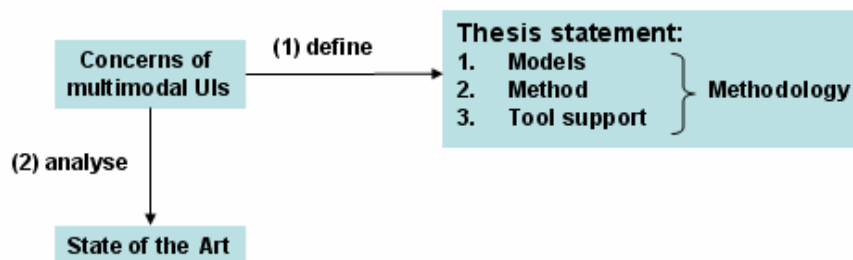


Figure 1-2 Benefits of the identified concerns of multimodal UIs

1.3 Terminology used in this thesis

In order to precisely identify the object of concern of this thesis, three fundamental terms often employed in the context of MM UIs are defined: mode, modality, media. The scientific community has now debated definitions and uses of these terms for more than twenty years without reaching clear consensus [Vand07]. For instance, the concepts of modality and multimodality mean different things to different stakeholders. In cognitive psychology, a modality denotes a human sense (e.g., vision, audition, taste, etc.) whereas in Human-Computer Interaction, multimodality corresponds more or less to interaction techniques that involve multiple human senses simultaneously. Much depends on the perspective, e.g., from a user or a system point of view, or on the degree of precision needed to solve or discuss a particular problem. In this section, we present our choices using a system perspective.

1. Introduction

1.3.1 Mode

The human body has five major senses which operate to gather information from the world around us (Figure 1-3): sight, hearing, smell, taste, and touch. Any stimulus to one of the sense areas is detected by sensory nerves and is sent to the brain for interpretation. The communication “mode” corresponds to the senses belonging to the motor and sensorial system of the user [Bell92] as it refers to the communication channel used by the two entities that interact [Schy05]. Consequently, two *input modes* exist that correspond to two motor and sensorial human systems: the oral mode from the hearing sense and the gesture mode from the touch sense. Similarly, five *output modes* correspond to the five senses: visual (sight), auditive (hearing), tactile (touch), olfactive (smell) and gustatory (taste) modes. By expanding this classification, four types of *input communication modes* are identified based on the implied sensorial system: graphical, vocal, tactile and gesture. Similarly, six *output communication modes* could be identified based on implied sensorial and motor systems: graphical, vocal, tactile, olfactory, gustatory, and gesture. A communication mode determines an interaction type between the user and the system. Thus, each communication mode has an associated interaction type. For instance, if the communication mode between the user and the system is graphical, the interaction is said to be graphical by analogy.

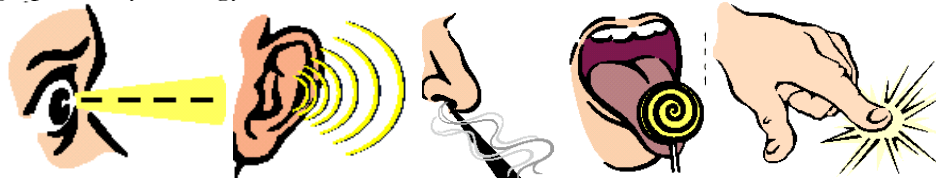


Figure 1-3 The human five senses

1.3.2 Media

Most of the authors agree in defining “media” as a technical support for information. In [Niga94], “media” is defined as a physical device that allows storing, retrieving or communicating information. Consequently, the definition is valid for all input devices (e.g., mouse, keyboard, microphone), for all output devices (e.g., screen, loud speakers) as well as for the devices storing the information (e.g., CD Rom, DVD) [Schy05]. Therefore, “media” is interpreted as being more than a “physical device” even if these two terms are used very often alternatively.

1.3.3 Modality

Regarding the term “modality”, Nigay’s definition [Niga97a] has been adopted because it clearly differentiates modalities by examining their intrinsic properties and because an extensive definition in terms of properties has been introduced in a meta-model of modalities [Bouc06]. The interaction modality is seen as a couple of a physical device d and an interaction language L : $\langle d, L \rangle$. A physical device is a system artifact that acquires (input devices) information (e.g., microphone, keyboard, or mouse) or delivers (output device) information (e.g., screen or loud speakers). An interaction language defines a set of conventional assembly of symbols that convey meaning (e.g., restricted natural language, direct manipulation, unrestricted natural language). The symbols are

1. Introduction

generated by actions applied on physical devices. According to this definition, typical examples of interaction modalities include:

- A graphical input modality described as the couple *<mouse, direct manipulation>*.
- A vocal input modality modeled as *<microphone, pseudo-natural language NL>*, where NL is defined by a specific grammar.
- A tactile input modality specified as the couple *<tactile screen, tactile commands>*.
- A graphical output modality modeled as the couple *<screen, graphics language>*.
- A vocal output modality described as the couple *<loud speakers, pseudo natural language NL>*.

1.3.4 Multimode, multimedia and multimodality

In this thesis, the definition of multimodality relies on a system-centered view. Thus, a MM system is a system having the capability to communicate with the end users through different types of communication modes and to extract and convey meaning automatically [Niga97c]. Thus, a *monomodal*, respectively *multimodal* system is referred to as any system that supports communication with the end user through a *single modality*, respectively *multiple modalities*. Multimodality refers to output as well as to input modalities: input, respectively output multimodal systems are employing at least two different input, respectively output, modalities.

Since the prefix “multi” implies the use of more than one suffix, a *monomedia*, respectively *multimedia*, system is referred to as any system that involves a single media, respectively multiple media. But multimedia systems also involve multiple types of communication modes. Consequently, what is the difference between a multimodal system and a multimedia system? A multimedia system allows the acquisition, the storage and the distribution of data, while a multimodal system is capable of acquiring and interpreting data, as well as storing and distributing these interpretations [Cout92]. Therefore, a multimodal system is a system with multimedia capabilities that enables semantic data handling.

Similarly, a *monomode*, respectively *multimode* system is any system relying on a single mode, respectively multiple modes, to support communication with the end user. A system can therefore be multimodal while being monomode (e.g., two modalities that are used in the same mode). Conversely, a multimode system subsumes its multimodality since at least two different modes are exploited.

Having defined these terms, we are now ready to define the central objective of this thesis and the working hypotheses it underlines.

1.4 Thesis

1.4.1 Thesis statement

In this thesis we argue that developing multimodal UIs is an activity that would benefit from the application of a **methodology** which is typically composed of: (1) a set of **models** gathered in an ontology, (2) a **method** manipulating the involved models and (3) **tools** that implement the defined method.

1. Introduction

Therefore, we will defend the following thesis statement:

Define a design space-based method that is supported by model-to-model colored transformations in order to obtain multimodal user interfaces of information systems from a task and a domain models.

The concepts introduced above are reviewed and defined in the next section.

1.4.2 Definitions of working hypotheses

1.4.2.a.1 The models

Model-based tools have been investigated since the late 1980's. The goal of these tools is to allow designers to specify the UI at an implementation independent level. The specification is usually shared between a set of components, called models, each model representing a facet of the interface characteristics. The number and type of these models is different from one approach to another. Our approach, for instance, considers the *task and domain models* since the initial design stage in order to encourage the user-centered design. Therefore, Chapter 3 will be dedicated to a precise description of the concepts involved in the considered models.

The *model-based approach* has been the target of some major criticisms [Myer00, Puer96, Shne06, Szek96]. The main shortcomings commonly cited are:

- (1) *High threshold*: the designers need to learn a new language in order to express the specifications of the UI.
- (2) *Low ceiling*: each model-based systems has strict limitations on the kind of UIs they can produce and the generated UIs are generally not as good as those that could be created with conventional techniques.
- (3) *Wide walls*: model-based systems do not support a wide range of possible explorations [Shne06].
- (4) *Unpredictability*: it is difficult to understand and control how the specifications are connected with the final UI. Therefore, the results may be unpredictable.
- (5) *Lack of propagation of modifications*: changes made to one model or to the final UI are generally not propagated to the other levels of specification.
- (6) *System dependent and private models*: a lot of models are strongly tied to their associated model-based system and can not be exported. Furthermore, some model specifications are neither publicly available, nor obtainable via a license.

Most of these problems could be addressed:

- (1) *High threshold*: most models can be built graphically in a design environment, which prevents users from learning the specification language. Even if the designers have to learn the specification language, the automation of a portion of the development should reduce the development effort.
- (2) *Low ceiling*: we believe that this criticism holds only for a specific kind of model-based generation tool, which generates the UI starting from very high level models

1. Introduction

(Task Model and/or Domain Model).

- (3) *Wide walls*: our approach considers a design space that benefits from a generative intrinsic quality. This enables designers to add design options or new values for the existing ones thus offering the possibility to extend the range of exploration.
- (4) *Unpredictability*: our approach relies on an explicit set of rules, fully documented and accessible. It offers the designer full control on the selection of those rules. The results of the application of a rule may be previewed.
- (5) *Lack of propagation of modifications*: although the problem of the impact of a modification made on a given model over the other models remains a tricky one, we will attempt to determine the side effects on the other models entailed by the application of a given rule.
- (6) *System dependent and private models*: we will make use of a UI description language publicly and freely available.

It is expected that the capabilities and the quality of automatically generated UIs and interactive applications will be expanding step by step and that in the future, perhaps a point will be reached where the capabilities of an interface builder as included in an Integrated Development Environment (IDE) and a Model-Driven Architecture (MDA)-compliant environment will become comparable.

The following definition was approved unanimously by 17 participants of the ORMSC plenary session meeting in Montreal on 23-26 August 2004.

The stated purpose of these two paragraphs was to provide principles to be followed in the revision of the MDA guide:

"MDA is an OMG initiative that proposes to define a set of non-proprietary standards that will specify interoperable technologies with which to realize model-driven development with automated transformations. Not all of these technologies will directly concern the transformation involved in MDA.

MDA does not necessarily rely on the UML, but, as a specialized kind of MDD (Model Driven Development), MDA necessarily involves the use of model(s) in development, which entails that at least one modeling language must be used. Any modeling language used in MDA must be described in terms of the MOF language to enable the metadata to be understood in a standard manner, which is a precondition for any activity to perform automated transformation."

Myers, Hudson, and Pausch [Myer00] argue that a model-based design tool will become successful from the moment that a low threshold and a high ceiling will be possible. A low threshold means that the designer or the developer does not need much to start developing a UI and that a simple UI could be obtained easily. In contrast, a high ceiling means that the tool has enough capabilities to produce sophisticated UIs while maintaining moderate the resources required for obtaining this UI.

Typically, UIs produced in interface builders and IDEs require some significant amount of resources (in terms of time, experience, skills), probably more than model-based IDEs, but their coverage is maximum (Figure 1-4): they exhibit a low threshold and a

1. Introduction

high ceiling. In contrast, first-generation model-based IDEs suffered from a high threshold and a low ceiling: they forced designers and developers to learn a new language (the one of the models), but once this effort is made, the resources required to produce the UI are low. However, only some limited UIs could be obtained. The second generation of model-based IDEs has expanded this coverage and the trend is now pursued by MDA-compliant softwares. It is worth to notice that such softwares are assumed to require less effort for learning the models since these models are already part of general purpose development methods like UML. We therefore hope that the coverage of such tools will progressively reach the coverage of traditional tools, but always with less resources involved.

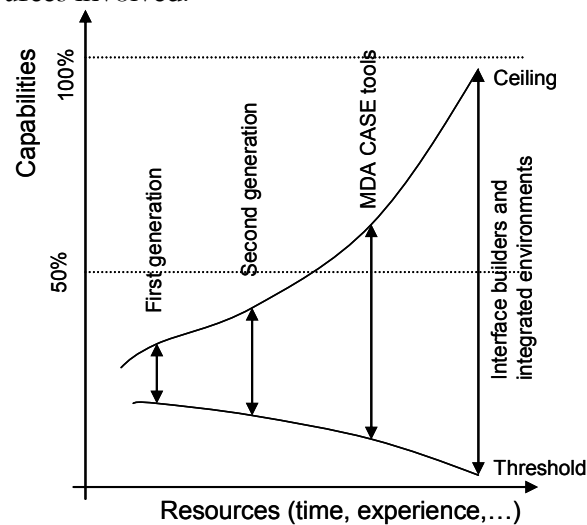


Figure 1-4 Capabilities vs. resources for producing a user interface

Model-based interfaces have also recognized advantages [Puer97]:

- (1) Advantages in terms of *methodology*:
 - It is a widely accepted software engineering principle to start a software development cycle with a specification stage [Ghez01].
 - The model-based approach supports a user-centred and UI-centred development life cycle: it lets designers work with tasks, users and domain concepts instead of thinking in engineering terms.
- (2) Advantages in terms of *reusability*:
 - In a multiplatform context, model-based tools can provide automatic portability across the different devices.
 - The availability of a complete description of the interface in a declarative form allows the reuse of some interface components.
- (3) Advantages in terms of *consistency*:
 - This approach ensures some form of consistency between the early phases of the development cycle (requirements analysis, specification) and the final product.
 - In a multiplatform context, it also guarantees a minimal consistency between

1. Introduction

the UI generated for different target platforms. This is not always possible when using traditional techniques where the development of each version of the UI is likely to be performed separately.

1.4.2.a.2 The method

The considered method consists of an *integrated approach* where all stages of the software development life cycle are covered in a principled way, from early requirements to prototyping and coding. This approach will benefit from a *design space* which will explicitly guide the designer in choosing values of design options that are appropriate to the MM UIs depending on parameters. In order to support these aspects our approach is also *transformational*, i.e. based on a catalogue of transformation rules. Similarly to the concept of schema transformation in database engineering [Hain02], we can define a transformation between source model \mathcal{M} and target model \mathcal{M}' as an operator which replaces a construct C in \mathcal{M} by a construct C' in \mathcal{M}' , or inserts a new construct into \mathcal{M}' , or removes an existing construct, while preserving a set \mathcal{P} of properties of \mathcal{M} . The set \mathcal{P} of properties we want to preserve includes:

- The *usability* of the UI.
- The *cross-platform consistency* of the whole information system, i.e. the consistency between the various versions of the UI.

1.4.2.a.3 The tool

Besides being model-based and transformational, our approach is also *computer-assisted* by automating, partially or totally, some repetitive tasks while offering some level of control to the designer. In order to conciliate computer-support and human control, we adopt a *semi-automatic approach* where:

- (1) Transformation rules are manually selected and parameterized by the designer, with a possibility to modify this configuration at any time.
- (2) Transformation rules are then automatically applied to reduce the design workload.

1.4.3 Scope

The current thesis basically concentrates on the following aspects:

- **Engineering of Interactive Systems** and in particular reactive systems that enable to interact with humans [Schy05]. On one hand, these systems imply that the inputs are not provided by another system, but by users whose behaviour cannot be predictable. On the other hand, reactive systems suppose that their outputs can be perceivable and easily interpretable by humans. Amongst these particular type of Interactive Systems we target **Information Systems** (ISs) defined as “a set of interrelated components that collect (or retrieve), process, store and distribute information” [Laud06]. This information is typically stored in databases. The importance of these ISs is vital in nearly all types of organizations. ISs can be distinguished depending on the level they serve in the organization (i.e., strategic, management or operational level) and on their major functional areas (e.g., sales

1. Introduction

and marketing, manufacturing and production, finance and accounting, and human resources). Typical examples of ISs (or subsystems) are a payroll system, a registration system or a sales order system. Examples of applications outside the category of ISs are entertainment applications, embedded systems or supervision systems.

- **Graphical, vocal and multimodal interaction** resulting from their combination. As specified in Section 1.3.1 the human body has five main senses to perceive outside stimuli. Of these senses, only three have been successfully used in Human-Computer Interaction. Sight and hearing are the most common modes of conveying information to a user. Touch has been used for silent vibration modes in mobile computing, but is not as common as the other two. Smelling and tasting output devices have been investigated and very few practical applications have been found interesting in an interaction context, because users find it impossible to rapidly perceive the information conveyed by these modes. For instance, smel-based interactions still found in their infancy [Kaye04, Brew06] show that olfactive feedback has been shown less effective than its graphical counterpart, but less disruptive [Bodn04]. Therefore, the former interaction remains less frequent in actual ISs. Apart from the basic senses, there are additional ones like thermo-reception or the sense of balance, but so far these cannot be used for interaction. Therefore, only sight and hearing are considered in this thesis as they are useful for information systems. By language abuse, we sometimes refer to interaction modalities (Section 1.3.3) by their corresponding communication mode.
- The methodology addresses the development of MM UIs for **predefined and constant contexts of use** specified at **design time**. Therefore any dynamic migration from one modality to another at run time is not supported.
- As our interest concerns the development of a general method for producing MM UIs based on a design space independent of the employed interaction modalities, the **fusion and fission aspects** of these interactions, although important, **will not be addressed**. Moreover, there are already a lot of research works dedicated to this particular area [Tour02], [Gait07], [Sun07].
- **The scope** of this work is **limited to multimodal UIs of IS**, which are **familiar to the vast majority of users and available on almost every platform**. Hence, we do not consider other families of UIs such as 3D UIs or tangible UIs.
- Consequently, other **aspects related to other layers of interaction application** (e.g., functional core, physical interaction) as they are defined by different **system architecture** (e.g., ARCH [Bass91], PAC-Amodeus [Niga94], W3C Multimodal Interaction Framework [Lars03b]) **are not addressed** in this thesis. In addition, **multimodal formal notations** such ICO [Nava06], SCXML [Barn08] or NiMMiT [Debo06] **are out of the scope of this thesis**.
- The primary goal of this thesis consists in defining a methodology that eases the design's workload when developing MMUIs. We **take for granted the benefits and shortcomings of these type of applications**. Therefore, **the question of**

1. Introduction

usability and accessibility of UI resulting from this methodology, although important, **will not be addressed explicitly** in this thesis.

- **The target audience** of this thesis is, on the one hand, the **HCI research community** and, on the other hand, **the professionals involved in the design and development of multimodal UIs**. In the remainder of this manuscript, we refer to these actors as “designers” or “developers”. The ultimate target is the end user for whom the benefit of MM UIs should become obvious.

1.5 Reading Map

The remainder of this thesis is structured according to Figure 1-5.

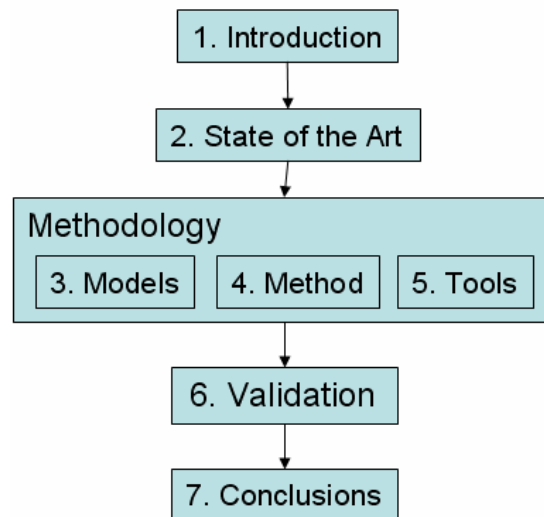


Figure 1-5 Thesis structure

Chapter 1 defines the thesis statement based on a set of concerns of MM UIs consider important for developing MM UIs. In addition, we have identified, defined and justified the terminology that will be further employed in this dissertation.

Chapter 2 is dedicated to the state of the art in the area of MM UIs. First, a description and a comparison between three significant conceptual MM frameworks are provided. Furthermore, the features of a set of UIDLs and MM UI development tools are detailed. We conclude with a summary of the state of the art that enables to establish a list of shortcomings of current UIDLs. Based on these shortcomings a set of requirements of MM UIs that argue the thesis statement are identified and will further be employed in the validation process of the results provided by our methodology.

Chapter 3 concerns the ontological aspects of our methodology. First, we justify the selection of the framework that will serve as a cornerstone of the thesis. Then, the composing models are detailed by emphasizing our conceptual contribution. Further, the

1. Introduction

semantics of our ontology is presented along with the supporting syntax and stylistics.

Chapter 4 is dedicated to the transformational method employed in the current thesis. The design space supporting this method and guiding the designer during the development process of graphical, vocal and MM UIs is defined, justified and detailed. Further, the selected graph-based transformational approach is expanded with the concept of colored transformation rules. The four steps of the transformational approach are identified and exemplified based on the design option composing the aforementioned design space.

Chapter 5 concerns the implementation aspects of our methodology. The tool supporting our method is introduced and each of the composing software modules are detailed by identifying their role in the corresponding transformational step.

Chapter 6 will address the external and internal validation of the methodology. The external validation consists of three case studies with different level of complexity: (1) an on-line polling system, (2) a car rental system and (3) a map browsing system. Further, we describe, analyse and interpret the results of an empirical validation with users thanks to a comparative study of MM UIs resulting from various designed options supported by transformations. For this purposes, three systems were employed: the second case study, a DVD rental system that is not described in the dissertation as it has the same level of complexity as the previous one and the map browsing system. The internal validation consists of reflections that aim to assess the characteristics of our methodology based on the set of considered requirements.

Chapter 7 concludes this dissertation by identifying its contribution to the three dimensions of the proposed methodology: models, method and tool implementation. In addition, the chapter presents several possible extension paths for future work and provides some personal reflexions with respect to the work presented in the current thesis.

1. Introduction

.

2 State of the Art

2.1 Introduction

After a survey of the research literature, the current chapter presents the state of the art issued from the real world MM UIs development solutions (Figure 2-1) that were considered to bring a significant contribution to the development of the methodology defined in Section 1.4.1. The considered aspects of the current chapter do not take into account MM related issues such as system architecture, fusion and fission mechanisms or MM formal notations that are out of the scope of this dissertation according to Section 1.4.3. Consequently, Section 2.2 provides a description of three conceptual MM frameworks and a comparison between them. In Section 2.3 the features of a set of eight UIDLs surveyed in the literature are presented along with their interest for our work. Further, Section 2.4 analysis some of the existing UI development tools considered important for us. The set of concerns identified in Section 1.2 are used to provide a set of features based on which the surveyed languages will be analysed in Section 2.5. As a result, a list of shortcomings will be identified so that to further help us establish the requirements addressed by the current thesis.

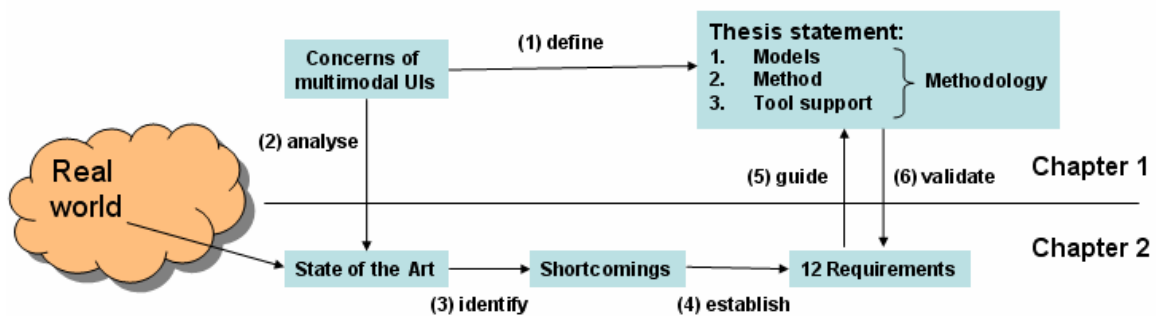


Figure 2-1 General schema for state of the art analysis

2.2 A structuring theoretical framework

This section details the features of three conceptual frameworks considered important for us, as they enable to manage different interaction modalities between the user and the system and the cooperation established between them. In conclusion we provide a comparison over the different points of view proposed by the frameworks.

2.2.1 TYCOON framework

The TYCOON (TYypes of COOperationN) framework holds an interest for our work as it enables to observe, evaluate and specify different types of cooperation among interaction modalities [Mart01].

In [Mart97] a modality is defined as a process which analysis and produces chunks of information. The TYCOON approach is based on the notions of *types* and *goals of cooperation* between modalities. As a result of a study made in domains such as Psychology, Artificial Intelligence, Human-Computer Interaction, five basic types of cooperation between modalities were distinguished:

- (1) **Transfer.** Specifies that a chunk of information produced by a modality is used by another modality. The transfer can appear either between two input/output modalities, or between an input and an output modality. The goals of this cooperation type are:
 - ✓ Translation: for instance, in hypermedia interfaces a mouse click generates the display of an image, or in information retrieval application, the user may express a request in one modality (e.g., speech) and get relevant information in other modality (e.g., video).
 - ✓ Improve recognition (e.g., mouse click detection may be transferred to speech modality in order to ease the recognition of predictable words (e.g., here, that).
 - ✓ Enable a faster interaction: when a part of an uttered sentence has been misrecognized, it can be edited using a keyboard so that the user doesn't have to type/utter again the whole sentence.
- (2) **Equivalence.** Two modalities are said to be equivalent if a chunk of information may be processed as an alternative, by either of the modalities. The goals of this type of cooperation are:
 - ✓ Improve recognition command: for instance, when a speech recognizer engine is not working accurately (e.g., in a noisy environment), the user can select the command with a stylus.
 - ✓ Adaptation to the user by customization: the user is allowed to select the modality he prefers.
 - ✓ Faster interaction: allows the system/user to select the fastest modality.
- (3) **Specialisation.** Indicates that a specific kind of information is always processed by the same modality. The goals of this cooperation type are:
 - ✓ Interpretation: the user is helped to interpret the events produced by the system.
 - ✓ Improve recognition: it enables an easier processing and it improves the accuracy of the speech recognizer since the search space is smaller.
 - ✓ Faster interaction: it decreases the duration of the integration and modality selection process.
- (4) **Redundancy.** Several modalities cooperate redundantly when they are processing the same information (e.g., the display of a confirmation dialog is replaced by two redundant user actions: typing "quit" and uttering "quit", thus enabling a faster interaction). Some benefits of redundancy have been observed:

2. State of the Art

- ✓ Support for users' natural acting: a case study revealed that sometimes users select their options (e.g., the town) both by speech and touch of tactile screen.
 - ✓ Increase of learnability: a redundant MM output involving both visual display of a text and speech utterance of the same text enables faster graphical interface learning.
- (5) **Complementarity.** Considers several modalities each one processing different chunks of information that are merged afterwards. The goals of this type of cooperation are:
- ✓ Faster interaction: as the two modalities can be used simultaneously and convey shorter messages better recognized than the longer ones.
 - ✓ Improve interpretation: for an expert the graphical output is sufficient, but for novice users a textual output is needed as well.

COMIT is a tool based on TYCOON framework that allows users to interact multimodality with the system in order to build GUIs. COMIT is defined by a command language which is used to specify several types of cooperation between speech recognition, keyboard and mouse interaction.

2.2.2 CARE properties

The CARE (Complementarity, Assignment, Redundancy and Equivalence) properties hold an interest for our work as it is a more advanced framework enabling to characterize the possible relationships occurring among different interaction modalities available in MM UIs. A modality is described as a couple of a physical device d and an interaction language L : $\langle d, L \rangle$ (Section 1.3.3). In order to give a formal definition of the CARE properties some parameters have been defined in [Cout95]:

- State: is a set of properties that can be measured at a particular time to characterize a situation.
- Goal: is a state that an agent intends to reach.
- Agent: is an entity capable of initiating the performance of actions (e.g., a user or a system).
- Modality: is an interaction method that an agent can use to reach a goal.
- Temporal relationship: characterizes the use over time of a set of modalities. The use of these modalities may occur simultaneously or in sequence within a temporal window, that is, a time interval.

Based on the above parameters, the following formal definitions of the CARE properties are specified:

- (1) **Equivalence (E).** Modalities of a set M are equivalent for reaching *state* s' from *state* s , if it is necessary and sufficient to use any of the modalities. M is assumed to contain at least two modalities:

$$\text{Equivalence}(s, M, s') \Leftrightarrow (\text{Card}(M) > 1) \wedge (\forall m \in M \text{ Reach}(s, m, s'))$$

E.g.: If we consider the following parameters:

- Modalities:

$$m1 = \text{speech input } \langle \text{microphone, restricted vocabulary-oriented natural language} \rangle,$$

2. State of the Art

$m2 = \text{written natural language} \langle \text{keyboard, command language} \rangle.$

- States:
 - $s = \text{a multimodal user interface with an unfilled text field widget,}$
 - $s' = \text{a multimodal user interface in which the text field widget from state } s \text{ is filled.}$
- Goal = reach stat s' from s .
- Agent = user.

Then an example of equivalent use of modalities is: the user can fill in the text field by employing any of the modalities $m1$ or $m2$.

- (2) **Assignment (A).** Modality m is said to be assigned to reach state s' from state s , if no other modality is used to reach s' from s :

$$\text{Assignment}(s, m, s') \Leftrightarrow \text{Reach}(s, m, s') \wedge (\forall m' \in M. \text{Reach}(s, m', s') \Rightarrow m'=m)$$

E.g.: If we consider the following parameters:

- Modality:
 - $m = \text{written natural language} \langle \text{keyboard, command language} \rangle.$
- States:
 - $s = \text{a multimodal user interface with an unfilled text field widget,}$
 - $s' = \text{a multimodal user interface in which the text field widget from state } s \text{ is filled.}$
- Goal = reach stat s' from s .
- Agent = user.

Then an example of an assigned modality is: the user can fill in the text field only by employing the modality m . No other modality can be used to reach the state s' .

- (3) **Redundancy (R).** Modalities of a set M are used redundantly to reach state s' from state s , if they have the same expressive power (they are equivalent) and if all of them are used within the same temporal window, tw :

$$\text{Redundancy}(s, M, s', tw) \Leftrightarrow \text{Equivalence}(s, M, s') \wedge (\text{Sequential}(M, tw) \vee \text{Parallel}(M, tw))$$

E.g.: If we consider the following parameters:

- Modalities:
 - $m1 = \text{speech input} \langle \text{microphone, restricted vocabulary-oriented natural language} \rangle,$
 - $m2 = \text{graphic input} \langle \text{mouse, direct manipulation} \rangle.$
- States:
 - $s = \text{a multimodal user interface with an unfilled combo box widget,}$
 - $s' = \text{a multimodal user interface in which the combo box widget from state } s \text{ is filled.}$
- Goal = reach stat s' from s .
- Agent = user.

Then an example of redundant use of modalities is: a combo box can be filled in by a user either by employing modalities $m1$ and $m2$ in parallel, or by using them sequentially but in the same temporal window (i.e., the user must act in a very short time interval so as the inputs can be treated as if they were parallel).

2. State of the Art

(4) **Complementarity (C).** Modalities of a set M are used in a complementary way to reach state s' from state s within a temporal window, if all of them must be used to reach s' from s , (i.e., none of them taken individually cannot cover the target state):

$$\begin{aligned} \text{Complementarity}(s, M, s', tw) &\Leftrightarrow (\text{Card}(M) > 1) \wedge (\text{Duration}(tw) \neq \infty) \wedge \\ &(\forall M' \in \mathbf{PM} (M' \neq M \Rightarrow \neg \text{REACH}(s, M', s'))) \wedge \text{REACH}(s, M, s') \wedge \\ &(\text{Sequential}(M, tw) \vee \text{Parallel}(M, tw)). \end{aligned}$$

E.g.: If we consider the following parameters:

- Modalities:

$m1 = \text{speech input} \langle \text{microphone, restricted vocabulary-oriented natural language} \rangle,$
 $m2 = \text{written natural language:} \langle \text{keyboard, command language} \rangle.$

- States:

$s = \text{a multimodal user interface with an unfilled text field widget allowing}$
 $\text{to input the name,}$

$s' = \text{a multimodal user interface in which the text field widget from state } s \text{ is filled.}$

- Goal = reach stat s' from s .
- Agent = user.

Then an example of complementary use of modalities is: modality $m1$ is employed by the user to utter his/her first name, while $m2$ is used to fill in the last name. None of the modalities taken individually can not be used to reach state s' .

ICARE (Interaction CARE) [Bouc04] is a component-based approach for the design and development of MM UIs, composed of elementary components. An elementary component supports a pure modality (e.g., speech only, graphics only). A graphical editor enables designers to graphically assemble the components according to the CARE properties. This assembly is afterwards transformed automatically into executable code. However, at run-time, this code is unable to adapt dynamically to the context of use. In addition, multimodality is limited to inputs.

2.2.3 W3C Multimodal Interaction Framework

The interest of our work in the W3C Multimodal Interaction Framework [Lars03b] identifies with its objectives:

- Identifying basic components of MM systems.
- Specifying markup languages used to describe information required by components.
- Ensuring data flowing among components.

The framework describes input and output modes widely used today and can be extended to include additional modes of user input and output as they become available. Figure 2-2 illustrates the basic components of the framework:

- End-user: enters input into the system and observes and hears information presented by the system.

2. State of the Art

- Input component: contains multiple input modes such as audio, speech, handwriting and keyboarding. EMMA [W3C04a] may be used to identify the semantics of data that represent the user's input.
- Output component: supposes multiple output modes such as speech, text, graphics, audio files and animation. The output component is supported by the following languages: SSML (Speech Synthesis Markup Language) used to describe how the words should be pronounced, XHTML, XHTML Basic or SVG used to describe how the graphics should be rendered and SMIL employed for the coordination of multimedia output.
- Interaction manager: is the logical component that coordinates data and manages execution flow from various input and output modalities. It maintains the interaction state and context of the application and responds to inputs from component interface objects and changes in the system and environment.
- Session component: provides an interface to the interaction manager to support state management and temporary and persistent sessions for MM applications.
- System and environment components: enable the interaction manager to find out about and respond to changes in device capabilities, user preferences and environmental conditions (e.g., which of the available modes the user wishes to use, the resolution of the display, if the display supports color or not).

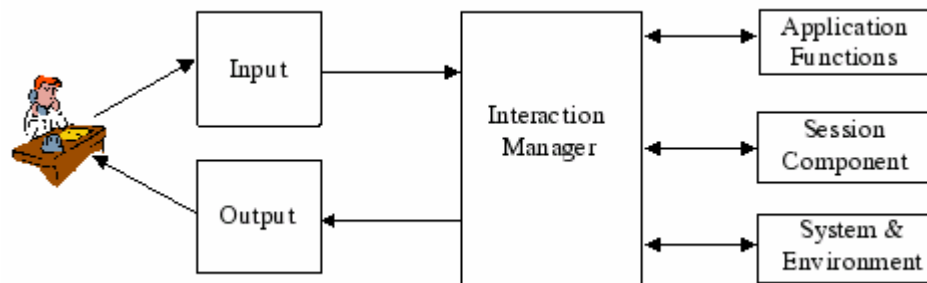


Figure 2-2 W3C Multimodal Interaction Framework

MM interaction requirements for MM interaction specifications are described in [Maes03]. Three increasing difficulty order levels for the management of input interaction are established:

- (1) *Sequential multimodal input*: corresponds to an input received from a single modality which may change over time. For this level it must be possible to specify which modality or device to use for input and hint or enforce modality switches.
- (2) *Simultaneous multimodal input*: implies that the inputs from several modalities are interpreted one after another in the receiving order, instead of being combined before interpretation.
- (3) *Composite multimodal input*: corresponds to an input received from multiple modalities at the same time and treated as a single, integrated compound input by downstream processes.

2. State of the Art

2.2.4 Comparison of theoretical frameworks

A first difference among the frameworks results from the way they are defining the notion of modality. While in TYCOON a modality is defined as a process which analysis and produces chunks of information, in CARE a modality is a couple of a physical device d with an interaction language L : $\langle d, L \rangle$. The W3C Framework defines modality as a type of communication channel used for interaction. The modality also covers the way an idea is expressed or perceived, or the manner in which an action is performed (e.g., voice, gesture, handwriting, typing).

Another difference encountered at the conceptual level is the existence of the *transfer* type of cooperation in TYCOON, concept that is missing in the case of CARE. Moreover, due to the fact that in several existing systems sounds are somehow specialized in notification errors (e.g., forbidden commands are signaled with a beep), in TYCOON a clear distinction of the type of specialization is being made:

- Modality-relative specialization: if sounds are used only to convey notification errors.
- Data-relative specialization: if errors only produce sounds and no graphics or text.

While CARE properties [Niga97b] define the relationships among devices and interaction languages, interaction languages and tasks, or among different modalities, in TYCOON the properties are used in a more restrictive way as they are describing only various types of cooperation among modalities. Another contrast concerns the manner of treating the interaction between the system and the user. With CARE it is possible to define cooperation between different modalities from both the system point of view (*system CARE properties*) and user's point of view (*user CARE properties*). The *user CARE properties* refer to the user's preferences that affect their choice for input modalities. With TYCOON only the system point of view is considered.

Some similarities can be identified among the frameworks. The *Redundancy* property defined in TYCOON and CARE frameworks could be expressed by employing modalities sequentially or in parallel which corresponds, respectively, to sequential and simultaneous MM input identified by W3C framework. Moreover, the *Complementarity* property supposes either a sequential or a parallel use of modalities treated as a single which corresponds, respectively, to sequential and composite MM input defined by the W3C framework.

2.3 User Interface Description Languages

This section presents a set of eight UIDLs surveyed in the literature that will further serve as a basis for identifying the shortcomings of the state of the art.

2.3.1 XISL

XISL (eXtensible Interaction Scenario Language) [Kats03] holds an interest for our work as it is the only web-based language that is supported by a tool enabling the development of MM UIs based on interaction scenarios between the user and the system.

2. State of the Art

The goal of XISL is to provide a common language supporting MM interaction that is characterized by three main features:

- Control dialog flow/transition: feature employed from VoiceXML
- Synchronize input/output modalities: feature employed from SMIL
- Modality-extensibility: ensured by XISL.

For this purpose, the language ensures the separation of the content (stored in XML/HTML files) from the interaction (described in XISL documents). This provides advantages in terms of: (1) reusability of the content and/or interaction, (2) improvement of specification's readability. Moreover, it supports the following types of cooperation between modalities: parallel input/output, sequential input/output, alternative input. The user, system or mixed initiative are supported by XISL for all the compliant devices: (i.e., PCs, mobile phones, PDAs). New devices could also be considered thanks to the use of non strict values of the elements specifying the input/output.

2.3.2 XIIML

XIIML (eXtensible Interface Markup Language) [Puer02a] represents an interest for our work as it provides a modality-independent level in the development life cycle from which final languages could be targeted. The main goal of the language is to enable a framework for the definition and interrelation of interaction data. Interaction data refers to the data that defines and links all relevant elements of a UI. From the structure point of view, XIIML language includes the following representational units:

- Components: organized collection of interface elements categorized in major interface components found in interface models:
 - ✓ User tasks: define a hierarchical decomposition of tasks in subtasks and the relationships between them.
 - ✓ Domain objects: is an organized collection of data objects and classes of objects that is structured into a hierarchy.
 - ✓ User types: categorized in a hierarchy of users.
 - ✓ Presentation elements: a hierarchy of interaction elements made of concrete objects which communicate with users.
 - ✓ Dialog elements: structured collection of elements that determine the actions available to the users.
- Relations: definition or statement that links two or more XIIML elements inside the same component or between different components.
- Attributes: features or properties of elements.

XIIML allows the development of UIs that must be displayed in a variety of devices. XIIML can be used to effectively display a single interface definition on any number of target devices. This is made possible by the strict separation that XIIML makes between the definition of a UI and the rendering of that interface which is left up to the target device to handle. There are a number of converters [Puer02b] used to transform a XIIML specification

2. State of the Art

to popular target languages (e.g., HTML, WML). XIML is also supported by a series of tools such as: XIML Validator, XIML Editor and XIML Viewer.

2.3.3 UIML

UIML [Abra04] is an XML-based language that holds an interest for our work as it provides: (1) a device-independent method to describe a UI, (2) a modality-independent method to specify a UI.

UIML allows describing the appearance, the interaction and the connection of the UI with the application logic. The following four key concepts underlie UIML:

- (1) *UIML is a meta-language*: UIML defines a small set of tags (e.g., used to describe a part of a UI) that are modality-independent, target platform-independent (e.g., PC, phone) and target language-independent (e.g., Java, VoiceXML). The specification of a UI is done through a toolkit vocabulary that specifies a set of classes of parts and properties of the classes. Different groups of people can define different vocabularies: one group might define a vocabulary whose classes have a 1-to-1 correspondence to UI widgets in a particular language (e.g., Java Swing API), whereas another group might define a vocabulary whose classes match abstractions used by a UI designer
- (2) *UIML separates the elements of a UI and identifies*: (a) which parts are composing the UI and the presentation style, (b) the content of each part (e.g., text, sounds, images) and binding of content to external resources, (c) the behavior of parts expressed as a set of rules with conditions and actions and (d) the definition of the vocabulary of part classes.
- (3) *UIML groups logically the UI in a tree of UI parts that changes over the lifetime of the interface*. During the lifetime of a UI the initial tree of parts may dynamically change shape by adding or deleting parts. UIML provides elements to describe the initial tree structure and to dynamically modify the structure.
- (4) *UIML allows UI parts and part-trees to be packaged in templates*: these templates may then be reused in various interface designs.

To create multiplatform UIs, concept 1 is used to create a vocabulary of part classes (e.g., a class *Button*) and concept 2 is used to separately define the vocabulary by specifying a mapping of the classes to target languages (e.g., mapping class *Button* to class *java.awt.Button* for Java and to the tag `<button>` for HTML 4.0). To create MM UIs, a multiplatform UI should be created and then each part is annotated with its mode (e.g., which target platforms uses that part). The behavior section from concept 2 is then used to keep the interface modalities synchronized. For example, it might be defined a UIML part class called *Prompt*, the mapping of *Prompt* parts to VoiceXML and HTML, and the behavior that synchronizes a VoiceXML and HTML UI to simultaneously prompt the user for input.

2. State of the Art

2.3.4 DISL

DISL (Dialog and Interface Specification Language) [Scha06] is a UIML subset that holds an interest for our work as it extends the language in order to enable generic and modality independent dialog descriptions.

Modifications to UIML mainly concerned the description of *generic widgets* and improvements to the behavioral aspects. Generic widgets are introduced in order to separate the presentation from the structure and behavior, i.e., mainly to separate user- and device-specific properties and modalities from a modality-independent presentation. The use of *generic widget* attribute enables to assign each widget to a particular type of functionality it ensures (e.g., command, variable field, text field, etc.). Further, a DISL rendering engine can use this information to create interface components appropriated to the interaction modality (i.e., graphical, vocal) in which the widget will operate.

The global DISL structure consists of an optional *head* element for meta information and a collection of templates and interfaces from which one interface is considered to be active at one time. Interfaces are used to describe the dialog structure, style, and behavior, whereas templates only describe structure and style in order to be reusable by other dialog components.

Current implementations of DISL language include media players application for playing mp3 files on mobile devices with limited resources or players run on PCs but controlled remotely from mobile phones.

2.3.5 VoiceXML

VoiceXML holds an interest for our work as it is the only standardized language [W3C04b] enabling vocal interaction extensively used in industry applications.

Its main goal is to provide web development and content delivery to voice applications, and to free the authors of such applications from low-level programming and resource management. It enables integration of voice services with data services using the traditional client-server paradigm. A voice service is viewed as a sequence of interaction dialogs between a user and an implementation platform. The dialogs are provided by document servers, which may be external to the implementation platform. Document servers maintain overall service logic, perform database and legacy system operations, and produce dialogs. A VoiceXML document specifies each interaction dialog to be conducted by a VoiceXML interpreter. User input affects dialog interpretation and is collected into requests submitted to a document server. The document server replies with another VoiceXML document to continue the user's session with other dialogs.

VoiceXML provides language features to support complex dialogs:

- Output of synthesized speech (text-to-speech)
- Output of audio files
- Recognition of spoken input
- Recognition of DTMF input
- Recording of spoken input

2. State of the Art

- Telephony features.

2.3.6 XHTML+Voice

XHTML+Voice, or X+V for short, holds an interest for our work as it is the only standardized web-based language [W3C04b] where traditional graphical interaction (i.e., keyboard, mouse) can be combined with vocal and tactile interactions (i.e., human finger, stylus pen).

The language is based on XHTML for graphical interaction, a simplified subset of VoiceXML for vocal interaction and XML events for synchronizing them. The three interactions available offer users the flexibility to select the modality that is the most suitable for achieving their tasks depending on the context (e.g., level of noise, availability of the hands). As X+V can afford a subset of the CARE properties (i.e., *Assignment*, *Equivalence* and *Redundancy* just for output), the user can combine the different interaction types available.

X+V applications can be developed either manually or by employing the IBM Multimodal Toolkit. The resultant specification is composed of: (1) graphical elements specifying the presentation and the behavior of the GUI, (2) vocal elements specifying the exchange of vocal information between the user and the system and (3) synchronization elements between the two previous elements. The graphical and vocal engine included in the multimodal browsers interpret separately the correspondent components. Currently, there are only two multimodal X+V browsers: Opera (Figure 2-3) and NetFront.

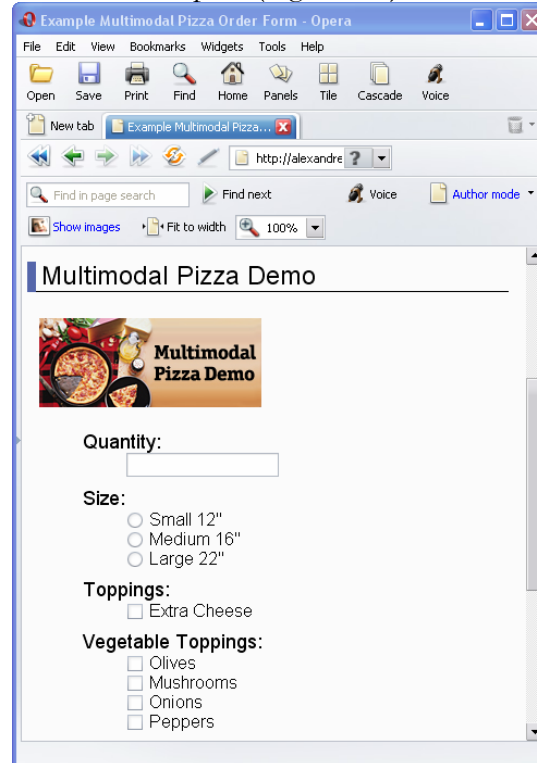


Figure 2-3 Multimodal X+V application interpreted with Opera browser

2. State of the Art

2.3.7 TeresaXML

TeresaXML holds an interest for our work as it is employed in a model-based, transformational approach for the development of MM and multi-device UIs.

The model-based approach [Mori04] is composed of the following steps: the initial task model for the envisioned system is transformed into a system task model that is specific to the target MM platform. The system task model is in turn transformed into an abstract UI, a concrete UI and then into the code of the final UI (i.e., X+V specifications).

2.3.8 EMMA

EMMA (Extensible MultiModal Addnotation Markup Language) holds an interest for our work as it is a markup language used to contain and annotate information automatically extracted from the input of users which manipulate MM UIs.

The language [W3C04a] is capable to convey meaning for different types of single input (i.e., text, speech, handwriting) and combinations of any previous modalities. These combinations are compliant with the W3C Interaction Framework (Section 2.2.3) (i.e., sequential, simultaneous and composite).

The language is used as a standard data interchange format between components of a MM system. EMMA is intended to be automatically generated by interpretation components used to represent the semantics (not directly authored by developers) of the users' inputs. The language does not represent a specification language and does not contain any transformational approach that initiates a progressive development from different models.

2.4 User interface development tools

This section provides the description of a set of monomodal and multimodal UI development tools considered important in the context of this dissertation.

2.4.1 Galatea Interaction Builder

Galatea Interaction Builder is a rapid-prototyping tool that supports XISL language [Kawa03]. It runs on PCs and can handle the following input modalities: speech, direct manipulation (mouse) and written natural language (keyboard) as well as output modalities such as: speech (text-to-speech), facial expression and graphic output. The tool provides a GUI design for domain-specific prototyping (Figure 2-4). The interaction scenario is presented under the form of a state transition diagram. Nodes of the diagram or MM interaction components, which correspond to XISL tags, are connected with links. The toolbar on the right side of the window provides the components used to specify the employed modalities (e.g., microphone for speech input, loud speaker for vocal output, a face symbolizing the output provided by an avatar).

2. State of the Art

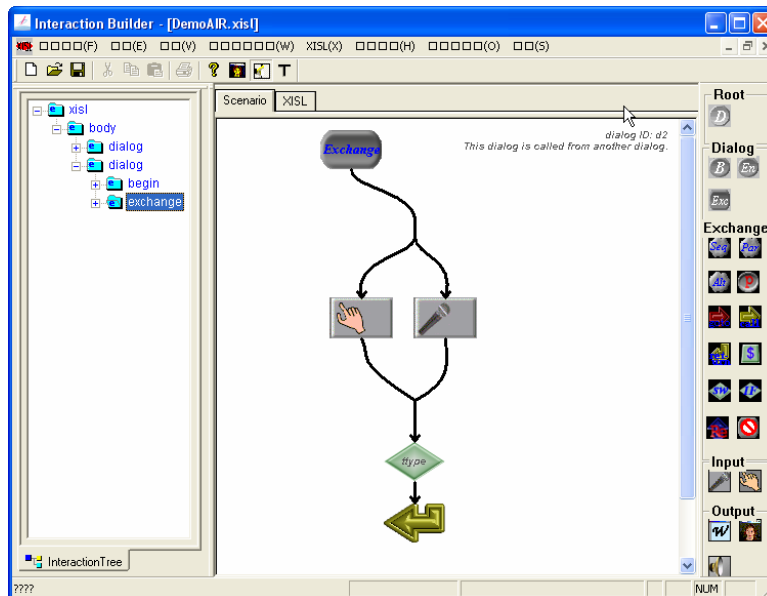


Figure 2-4 The Interaction Builder graphical user interface

2.4.2 UIML Development Toolkit

UIML Development Toolkit (Figure 2-5) provides support for the UIML language by allowing designers to generate high fidelity interfaces and production code. The tool is a plug-in for the Eclipse IDE and is supported by LiquidUI, a tool that integrates a set of converters for different software platforms (e.g., Java, HTML, WML, VoiceXML, C++).

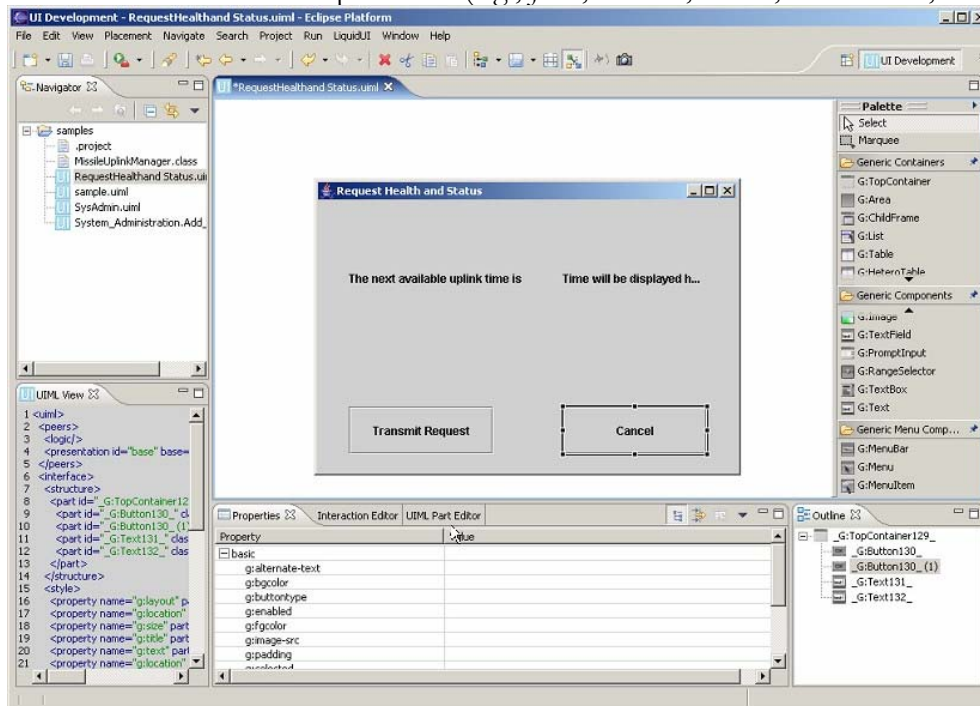


Figure 2-5 UIML Development Tool

2. State of the Art

2.4.3 WebSphere Voice Toolkit

IBM WebSphere Voice Toolkit supports the VoiceXML language and offers one of the most complete set of features required to deploy vocal-based applications. Powered by Eclipse technology, the toolkit eases the development of VoiceXML applications as it does not require in depth knowledge of voice technology. It offers a full-featured voice development environment including: (1) Graphical communication flow builder (Figure 2-6), (2) VoiceXML development and debugging, (3) Grammar development and debugging, (4) Pronunciation Agent builder, (5) Call Control extensible Markup Language (CCXML) development.

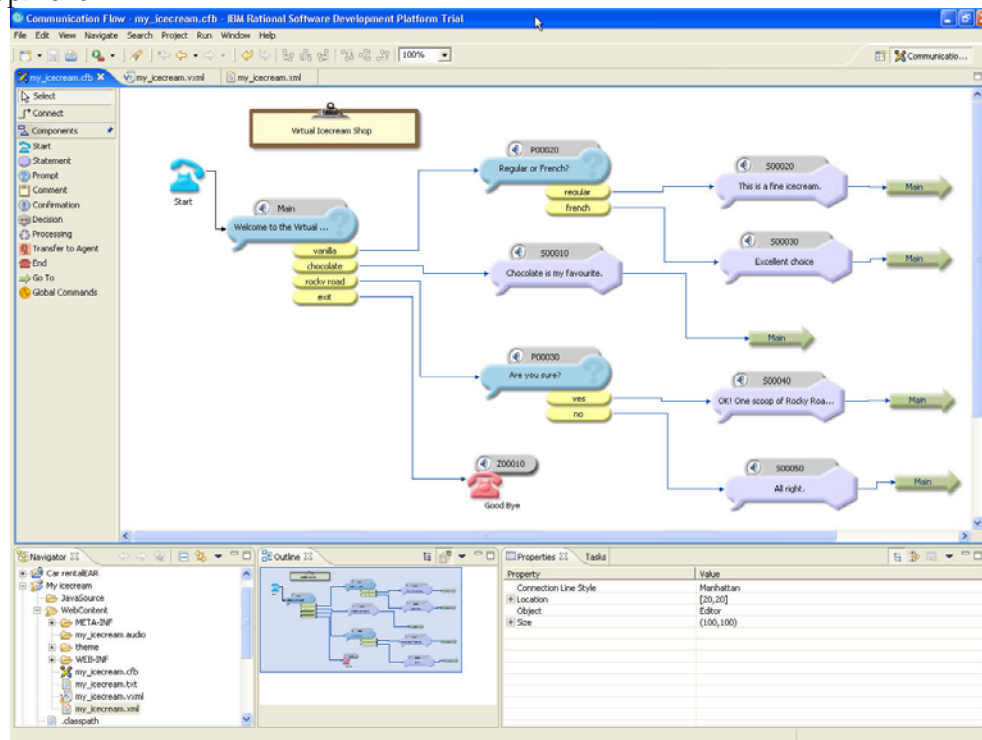


Figure 2-6 IBM WebSphere Voice Toolkit – communication flow builder perspective

2.4.4 Teresa

Teresa (Figure 2-7) is a transformation-based environment that supports the development of MM UI in TeresaXML language according to the steps identified in Section 2.3.7. However, the transformation process uses parameters that are not related into a coherent and explicit set of design options. In addition, Teresa transformations are hard coded and embedded into the code.

2. State of the Art

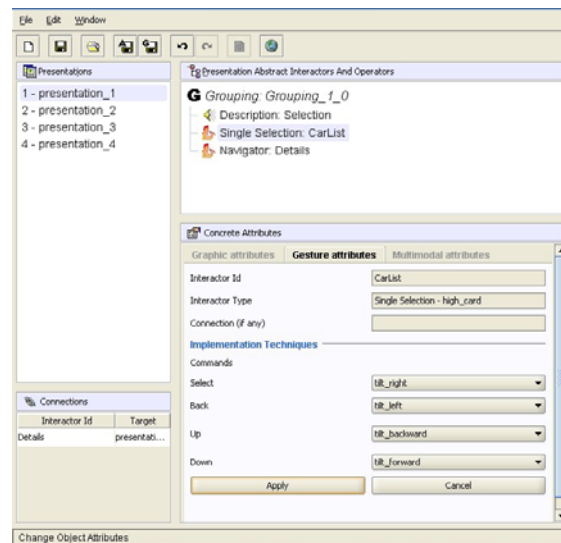


Figure 2-7 Authoring a multimodal UI with Teresa

2.4.5 MONA

MONA (Mobile multimodal Next generation Applications) [Aneg04] holds an interest for our work as it is a complete environment for producing web-based MM applications (Figure 2-8).

The tool involves a presentation server for a wide range of mobile devices using wireless LAN and mobile phone networks that generates graphical or MM (i.e., graphical and vocal) UI able to dynamically adapt to different devices: WAP-phones, Symbian-based smart phones or PocketPC and PDAs. The application design process is based on use cases that allow, for each device, the refinement and validation of the design of MM UI prototypes. These prototypes are further submitted to a heuristic evaluation performed by evaluators with design experience.

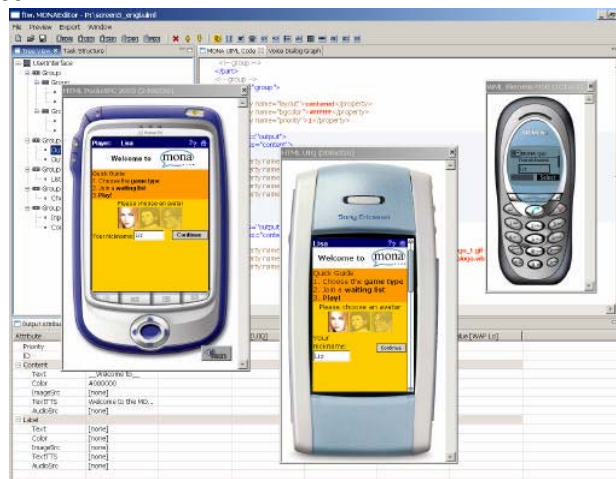


Figure 2-8 MONA editor with real time GUI previews

2. State of the Art

2.4.6 SUEDE

SUEDE holds an interest for our work as it is a speech interface prototyping tool that enables rapid and iterative creation of prompt-response vocal interfaces [Anno01].

SUEDE couples a simple prompt/response card model with the Wizard of Oz technique. There are four types of cards: start card, prompt card, response card and group card. The Wizard of Oz technique enables unimplemented technology to be evaluated by using a human to simulate the response of a system. Wizard of Oz methodologies have a long tradition in the design of vocal systems as well as the ability to suggest functionality before the implementation of the system. The Wizard simulates dialog transition as a computer would, reads the system prompts to the participants and process their response.

The iterative steps supported in SUEDE are: design, test and analysis. In design mode (Figure 2-9), the speech designer begins to create dialog script examples. After constructing several scrip examples, the designer begins to construct a design graph that represents a more general design solution. In the test phase, the designer tries out a design with target users. Due to the fact that the wizard recognizes user's responses, no speech recognition or speech synthesis is necessary to test Suede prototypes. During the analysis, designers examine collected test data, deciding how this should influence the next design iteration in order to obtain a more appropriate flow of the UI.

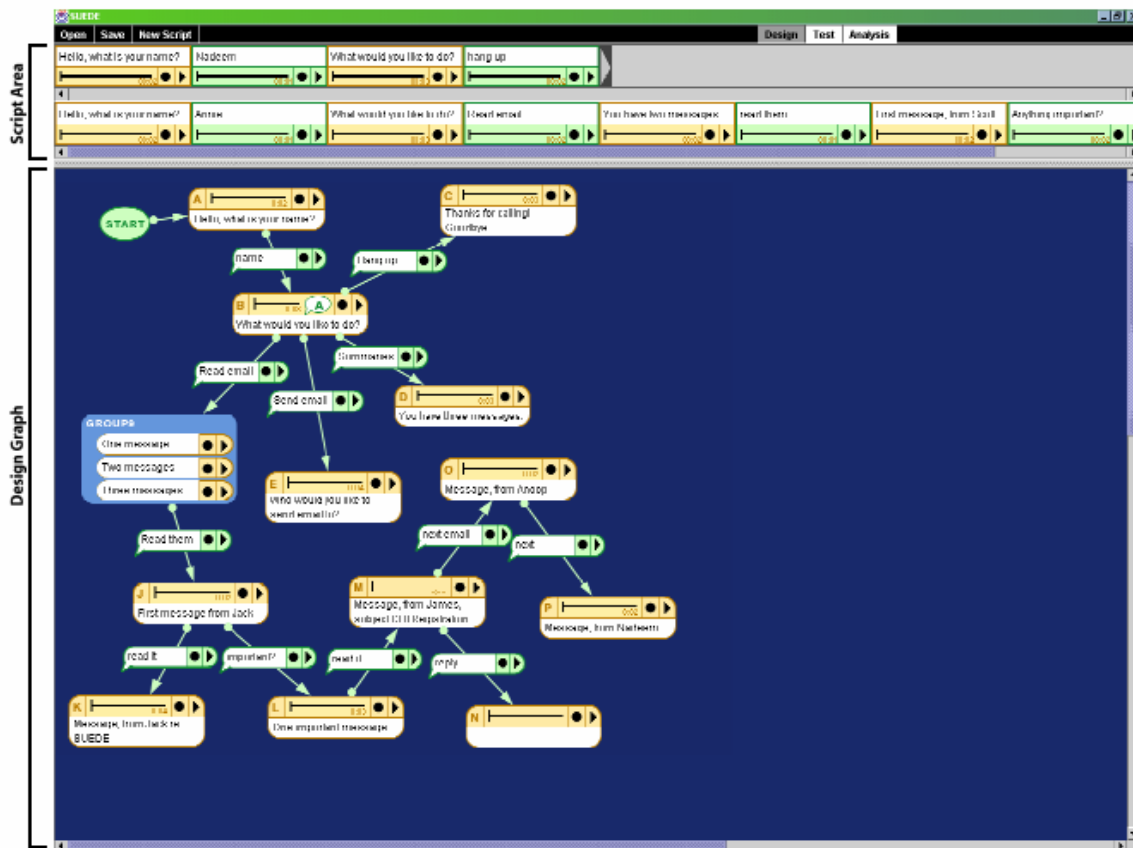


Figure 2-9 Design mode in SUEDE

2. State of the Art

2.4.7 CSLU Toolkit

CSLU Toolkit holds an interest for our work as it provides a basic framework and the tools to build, investigate and use MM applications involving the following capabilities: speech recognition, natural language understanding, speech synthesis and facial animation technologies.

The toolkit is used for developing applications in Tcl/Tk and C programming languages:

- *RAD (Rapid Application Developer)*: is an easy to use graphical authoring tool (Figure 2-10) that enables the creation of structured dialogues applications and a wide variety of interactive programs that run both over the telephone and on desktop PCs. RAD component allows to drag and drop dialogue states onto a canvas, interconnect them together, and configure them to play audio files, create animated text-to-speech, recognize spoken language or display images.
- *Baldi*: is an animated, anatomically correct head that can be used from within RAD and in other applications to provide a synchronized visual speech source. It allows the configuration of many aspects of the face and the saving of these customized configurations for later use.
- *Baldi Sync*: allows users to record a phrase and then animate Baldi with the user's voice.
- *Festival*: is the text-to-speech component of the toolkit.

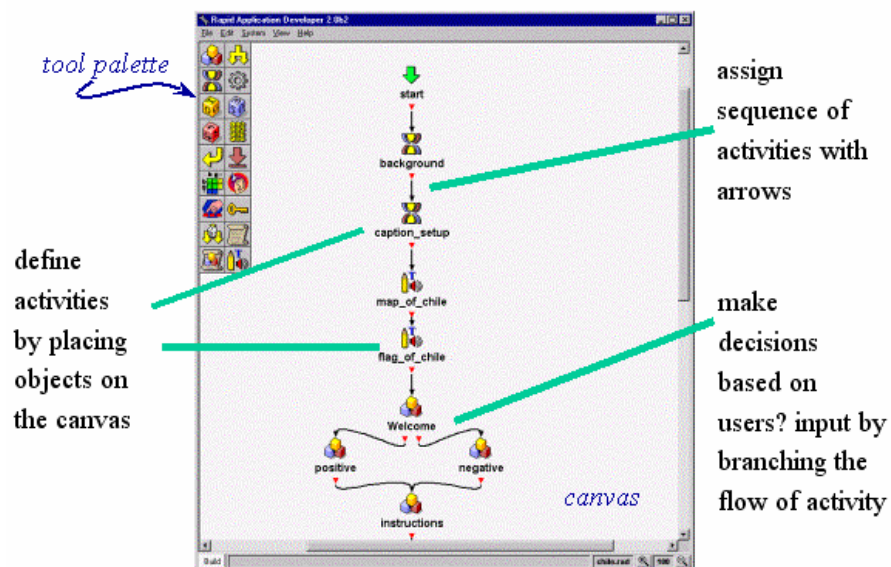


Figure 2-10 CSLU toolkit - the graphical authoring editor

2.4.8 MOST

MOST (Multimodal Output Specification Platform) platform [Rous05] holds an interest for our work as it enables the design of output MM systems (i.e., graphical, vocal and tactile modalities) based on a three-step process: analysis, specification and simulation.

2. State of the Art

In the analysis phase the output interaction components (i.e., mode, modality and medium) are identified (Figure 2-11). The specification phase formalizes the results of the previous phase based on a series of attributes and criteria assigned to each specific output interaction component. Depending on the current state of the interaction context, a behavioral model allows the identification of the most suitable output form that can be further used in order to present each interaction component. The behavioral model is composed of a set of selection rules that produces the appropriate MM presentation. Finally, the simulation phase is based on the *WWHT* conceptual model which aims to answer the following questions:

- What is the information to present?
- Which modality/modalities should be used to present this information?
- How to present the information using this/these modality/modalities?
- Then, how to handle the evolution of the resulting presentation?

This model is supported by a tool that enables to develop a prototype of the complete system as well.

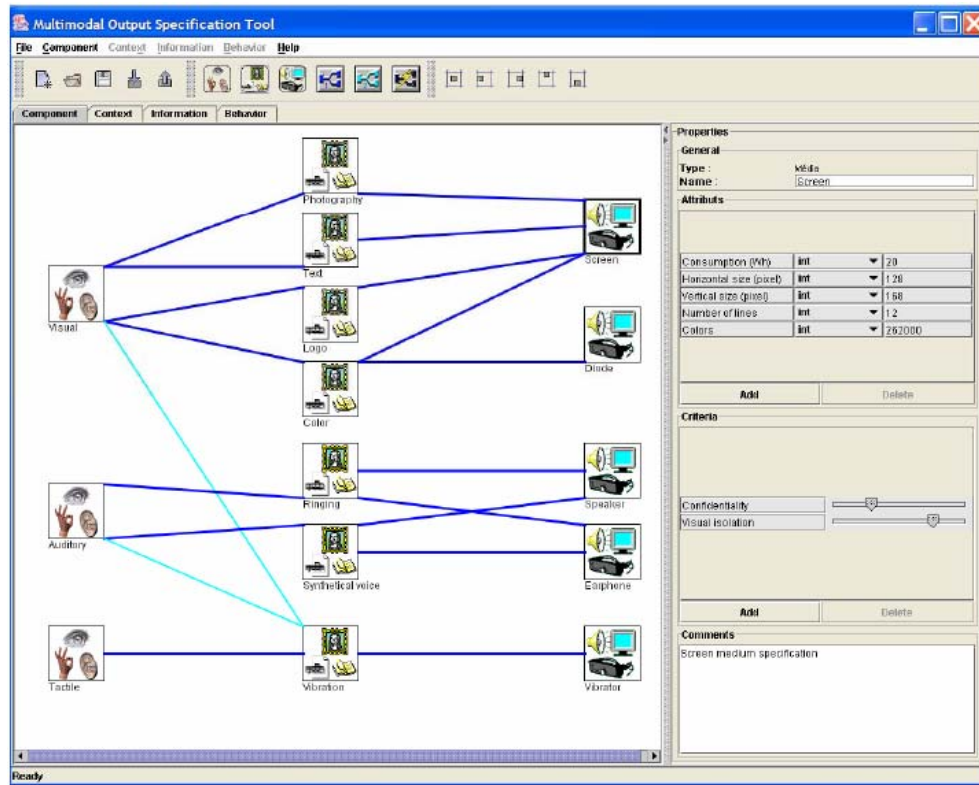


Figure 2-11 Interaction component editor in MOST

2.4.9 OpenInterface Platform

OpenInterface [Open07] holds an interest for our work as it aims to provide an open source platform for the design and rapid development of MM prototyped applications as a central tool for an iterative user-centered process.

2. State of the Art

The basic objects manipulated by the OpenInterface platform are called components (Figure 2-12). Each one represents a bundled piece of software that provides a set of services/functionalities ranging from input devices driver, signal-treatment algorithm, network module, graphical interface, etc. To be able to manipulate a component, the OpenInterface platform requires the description of the component's interface. This description is specified in CIDL (Component Interface Description Language). Once the CIDL is specified, the component can then be reused easily in any OpenInterface application. OpenInterface components can be composed together to create a network of components managing some advanced task. Such an inter-connection of components is called a pipeline. In order to be manipulated by the OpenInterface platform, a pipeline must be specified in the PDCL (Pipeline Description and Configuration Language). A PDCL description defines the components that are used in the pipeline and the way they are interconnected. The platform benefits from a set of advantages:

- It allows seamless integration of heterogeneous software. The platform manages the translation/communication of the data among the different programming languages using existing tools. The currently supported languages are C/C++, Java and Matlab, but support of other languages can be added rather easily.
- It allows rapid prototyping of MM applications thanks to the bundled generic fission and fusion mechanism and the easy software connection.
- The delivered software is a reusable independent unit.

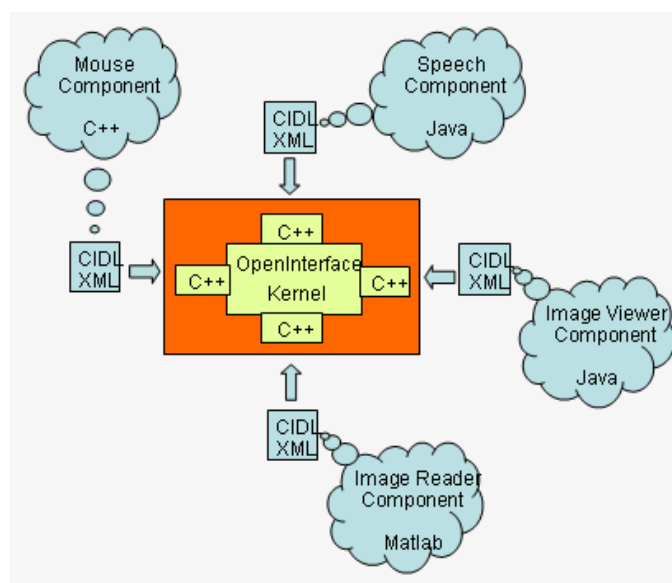


Figure 2-12 Integration of heterogeneous components in OpenInterface

2.4.10 A Toolkit of Multimodal Widgets

The toolkit holds an interest for our work as it aims to ease the development of MM UIs by fulfilling the following four requirements [Crea00]: (1) each widget should be capable of producing feedback in multiple modalities with no preference given to any particular

2. State of the Art

modality, (2) the widgets should be capable of using the most suitable modality or limiting the use of a modality which has reduced resources, (3) it should be easy to change the feedback produced by a widget in one or more modalities without any effect over the rest of the modalities, (4) the produced feedback should be consistent, both between widgets and between modalities.

These requirements are inferred from observations made over HCI that might be different depending on the context in which the interaction is taking place (i.e., indoor/outdoor, noisy/quiet environment, alone/a group). Therefore, the authors of the toolkit identify the necessity of conveying the interfaces to the users by employing different output modalities, referred here as sensory modalities (i.e., all auditory output is one sensory modality and all visual output is another modality).

Figure 2-13 shows the architecture of the toolkit. The *feedback controller* ensures requirement (1). It translates the external events into requests for feedback independent of the modality, which are further transmitted to the modality mapper. The *resource manager* ensures requirement (2). It receives the input from three sources: the control panel that allows the users to set the weight for a particular modality, the output modules that indicate if the resources are sufficient to render the widgets in a particular modality taking into account the weight set by the user, and the external applications that can use the resource manager's API to influence the weight of different modalities. Requirement (3) is ensured by both the *output modules* and the *control panel*. Because the widget behaviour does not encapsulate the feedback given by the widget, it is simply a matter of changing the feedback of the widgets. To replace one feedback with another, a simple switch between the existing output module and another module should be operated. To supplement the existing feedback with another one in a different modality, a new output module should be added to the toolkit. For all widgets, any option set in the control panel is added to the request made for the feedback in the *modality mapper*. There is a modality mapper for each output module the widget uses. The *rendering manager* ensures requirement (4). It detects if a widget's feedback clashes (e.g., two similar sounds that are being played at the same time, thus interfering with each other and rendering the conveyed information unintelligible) with the feedback from other widget and suggest a change in the feedback.

2. State of the Art

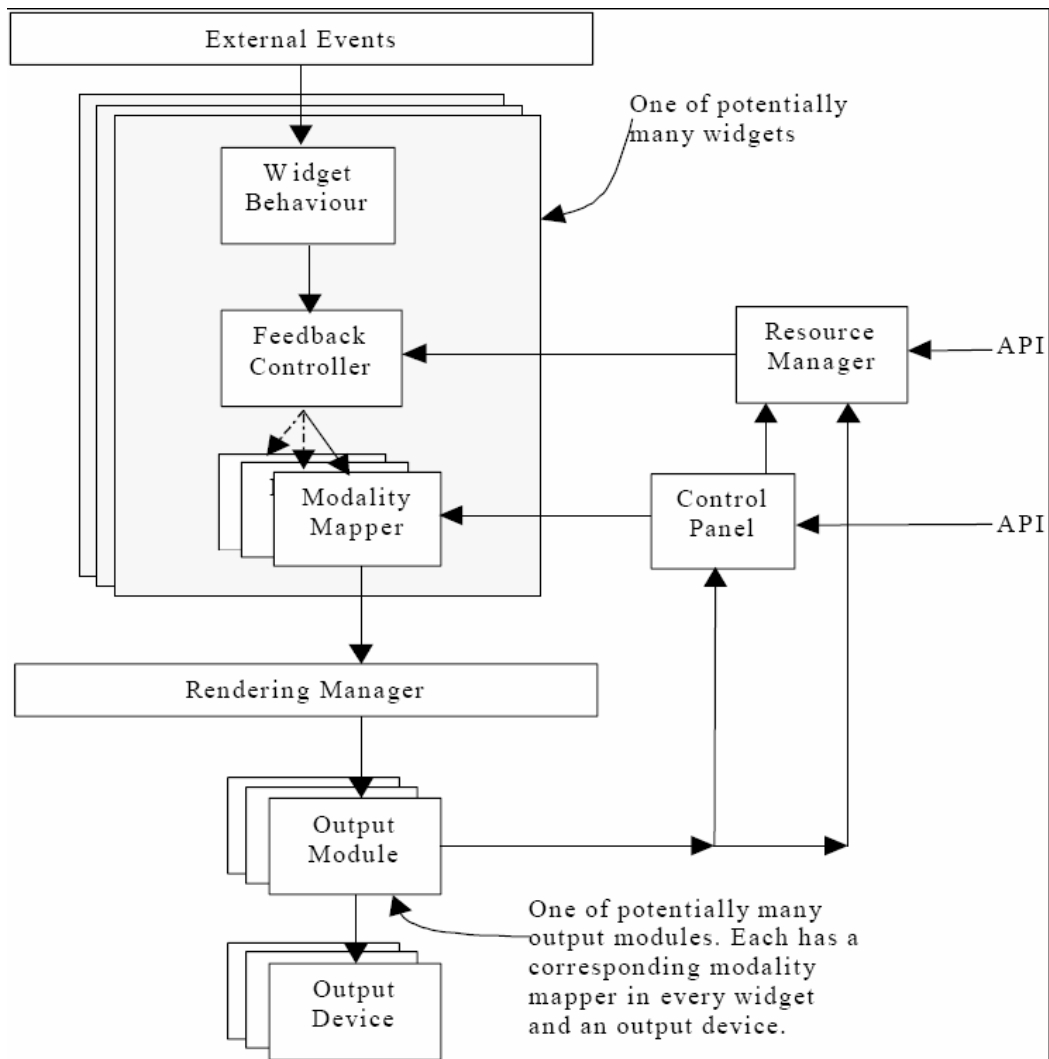


Figure 2-13 Multimodal toolkit architecture

A concrete example of how the toolkit can be used for a standard button is illustrated in Figure 2-14. The programming language offered by the toolkit is very close to Java Swing so that the knowledge overload of the developers is practically unexistent as the MM rendering of the UIs is ensured by the system. Currently, the toolkit has been implemented with two widgets, a button and a progress bar using two modalities: graphical and audio.

2. State of the Art

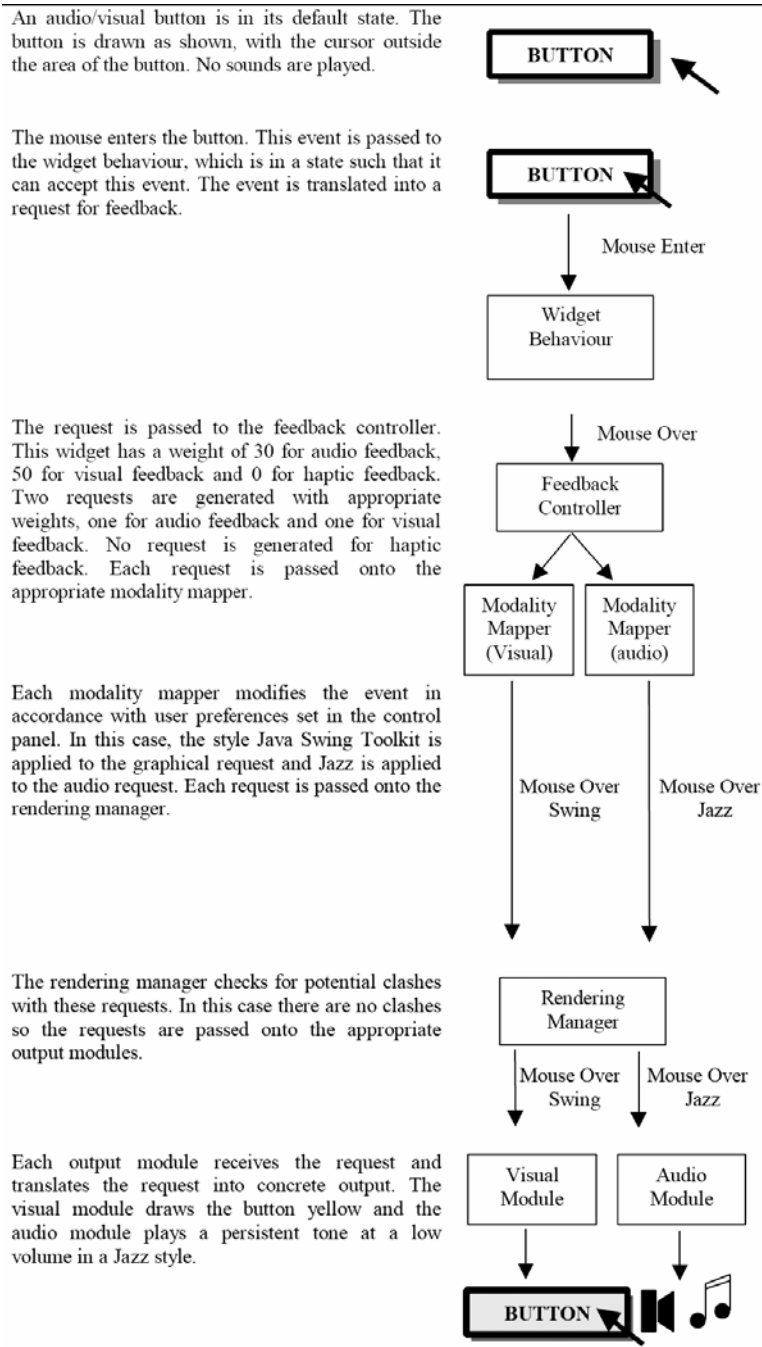


Figure 2-14 The toolkit architecture –button feedback to mouse-over event

2.4.11 FRUIT

This system holds an interest for our work as it separates the traditional widgets in two classes[Kawa96]: *abstract widgets* that are used to manage the semantical features and *concrete widgets* that are employed for rendering purposes in graphical and vocal modalities. This

2. State of the Art

separation is the result of the observations made over the development of current systems where widgets manage both the semantical and presentational aspects of a UI.

The abstract widgets are classified in three main classes:

- *Base abstract widgets*: basic unit interaction objects (e.g., pushbutton, text entry).
- *Container abstract widgets*: objects that organize abstract widgets and control focusing policy.
- *Compound abstract widgets*: a group of abstract widgets which behave in a specific way (e.g., file selection boxes are composed of several widgets and may have specific interaction protocol).

The base widgets are sub-grouped in four main classes: (1) *command*: usually rendered in GUIs as push buttons, they can also be keyboard entries or uttered commands in VUIs, (2) *selection*: rendered in GUIs as list boxes, group of radio buttons, menus, whereas in VUIs they can be concretized in words and numbers in VUIs, (3) *valuator*: rendered in GUIs as slidebars, (4) *TextDisplay*, *TextInput*: rendered in GUIs as labels, whereas in VUIs they support the any vocal output system or user input. The abstract widget container is grouped in three main classes as well: (1) *shell*: rendered in GUIs as top level widgets (i.e., usually the windows), (2) *menu*: is a type of shell that takes temporarily the focus, (3) *group*: manages the focus dispatching among the contained abstract widgets.

A FRUIT system is composed of three parts (Figure 2-15):

- The rendered widgets are dispatched in one or multiple *interaction shells*. Usually, the designer chooses a single shell (e.g., vocal), but multiple shells can be triggered (e.g., graphical) for completion purposes.
- The abstract widgets centralize the *application logic*. They provide an interpretation at the application level of the operations triggered over the rendered widgets.
- The *session manager* runs on each host as a daemon to manage FRUIT applications.

The designer develops the UI by manipulating the abstract widgets. The end-user interacts with the rendered widgets in one or multiple interaction shells (e.g., graphical, vocal). The rendered widgets communicate to the abstract widgets the operations to trigger via an interaction protocol. The session manager component manages the FRUIT application. One of the shortcomings of the system consists of the fact that the choice for the presentation of a widget belongs to a black box that takes the decision depending on the assigned abstract widget in the interaction shell. Thus, the designer's decision is practically inexistent as the choice of the interaction object belongs entirely to the system.

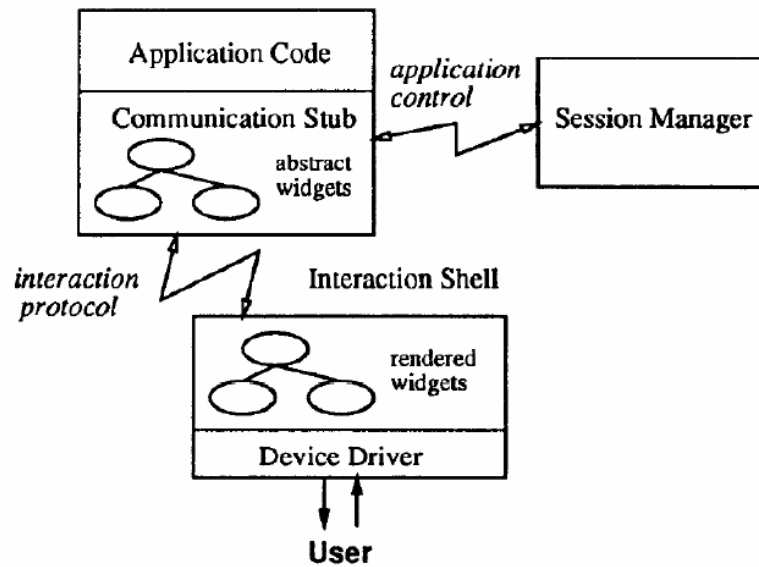


Figure 2-15 The architecture of a FRUIT system

2.5 Conclusion

2.5.1 Summary of the state of the art

As illustrated in Figure 1-2 a set of features that will enable to determine the constraints of the MM languages surveyed in the current chapter (Table 2-1) are inferred from the concerns regarding MM UIs identified in Section 1.2:

- *Input modalities*: specify the input modalities that can be employed by the end-user while interacting with the system. As there is a real need for more modality interaction flexibility that enables users to select the most suitable one for their task (**Concern 1**) the employed input modalities are identified:
 - ✓ Graphical: specifies the interaction devices (e.g., keyboard, mouse).
 - ✓ Vocal: specifies the type of vocal input (e.g., speech recognition).
 - ✓ DTMF (Dual Tone Multi-Frequency): is the system used by the touch-tone telephones that consist in assigning a specific frequency to each key so that it can easily be identified.
- *Output modalities*: specify the output modalities employed by the system when providing information to the users. This feature is also inferred from **Concern 1** as multiple modalities should be made available so as to enable the interaction flexibility in output:
 - ✓ Graphical: specifies the output device (e.g., PC screen, GSM screen).
 - ✓ Vocal: specifies the type of output (e.g., speech synthesis, text-to-speech).
 - ✓ Avatar: is an animated face that behaves like humans; it is endowed with gesture features and is able to make speech conversation with humans.

2. State of the Art

- *Separation of modalities*: specifies if the language specifications for the involved modalities are separated or combined (**Concern 2**).
- *CARE properties support for input modalities*: specify which of the CARE properties are supported for input modalities. This feature is inferred from **Concern 3** (the CARE properties are used to identify and characterize the different types of modality combinations) and from the need to enhance the device effectiveness.
- *CARE properties support for output modalities*: specify which of the CARE properties are supported for the output modalities. By analogy with the previous paragraph this feature is inferred from **Concern 3** and from the need to enhance the device effectiveness.
- *Independence of modality*: specifies the existence in the development life cycle of a modality-independent level for language specification (**Concern 4**).
- *Extensibility for new modalities*: identifies if the language enables to be extensible with new input and output modalities (**Concern 5**).
- *Design options*: identifies the existence of design options in the development process of UIs (**Concern 7**).
- *Model-to-model transformational approach*: indicates the existence of a transformational approach between the models involved in the development process. This feature is motivated by the need to provide a model-driven engineering approach where model-to-model and model-to-code transformations are applied in order to produce the final MM UI.
- *Development tools*: specifies the name(s) of the development tool(s). This feature is motivated by the reduced number of MM UIs which might be due to the lack of development tools. It allows to investigate the existence of these tools enabling the automatic development of MM UIs.
- *Interpretation/Rendering/Converter tools*: identifies the name(s) of the interpretation/rendering tools. Some languages converters were developed to target already standardized languages. This feature is justified by the need to investigate the existence of tools that could interpret/render/convert the eventually target languages of our methodology.

2. State of the Art

Language / Features	XISL	XIML	UIML + DISL	VoiceXML	X+V	TeresaXML	EMMA
Input modalities	Graphical ✓ keyboard ✓ mouse ✓ touch screen Vocal ✓ speech recognition DTMF	Graphical ✓ keyboard ✓ mouse DTMF	Graphical ✓ keyboard ✓ mouse Vocal ✓ speech recognition DTMF	Vocal ✓ speech DTMF	Graphical ✓ keyboard ✓ mouse ✓ stylus pen ✓ touch screen Vocal ✓ speech recognition	Graphical ✓ keyboard ✓ mouse ✓ stylus pen ✓ touch screen Vocal ✓ speech recognition DTMF	Graphical ✓ keyboard ✓ mouse Vocal ✓ speech recognition
Output modalities	Graphical ✓ PC screen ✓ PDA screen Vocal ✓ speech synthesis ✓ text-to-speech ✓ audio Avatar	Graphical ✓ PC screen ✓ GSM screen	Graphical ✓ PC screen ✓ GSM screen Vocal ✓ speech synthesis ✓ text-to-speech ✓ audio	Vocal ✓ speech synthesis ✓ text-to-speech ✓ audio	Graphical ✓ PC screen ✓ handheld devices screen Vocal ✓ speech synthesis ✓ text-to-speech ✓ audio	Graphical ✓ PC screen ✓ handheld devices screen Vocal ✓ speech synthesis ✓ text-to-speech ✓ audio	-
Separation of modalities	No	-	Yes	-	Yes	Yes	-
CARE properties support for input modalities	A, E	-	A, E	-	A, E, R	A, E, R	-
CARE properties support for output modalities	A, E, R	-	A, E, R	-	A, E, R	A, E, R	-

2. State of the Art

Independence of modality	No	-	Yes	-	No	Yes	-
Extensibility for new modalities	Yes	No	Yes	No	No	Yes	Yes
Design options	No	No	No	No	No	Yes	No
Model-to-model transformational approach	No	Yes	No	No	No	Yes	No
Development tool	Galatea Interaction Builder	XIML Validator, Editor, Viewer tools	UIML development tool	IBM WebSphere Voice Toolkit	IBM Multimodal Toolkit	Teresa	No
Interpretation/Renderer/Converter tools	Internet Explorer 6 with multimodal software support components, Anthropomorphic spoken dialog agent toolkit	Converters to HTML, WML	LiquidUI (converter for HTML, WML, VoiceXML, Java, etc.)	IBM VoiceXML browser	Opera browser, NetFront browser	Teresa (generation of X+V specification)	No

Table 2-1 Comparison of the surveyed user interface description languages

2.5.2 Shortcomings

We identified the following shortcomings that serve as incentives to consider this topic an important, original, yet unsolved and challenging research problem by observing the current practice of MM UI usage:

- *Shortcoming 1. Lack of a fast interaction:* the different input/output monomodal interactions enabled by most of the current applications hinders users to take benefit of their natural multimodal interaction skills. Therefore, the users are slowed down when responding/accessing to the delivered information.
- *Shortcoming 2. High incidence of errors and difficult error recovery:* even if multiple interactions are available, there is a lack of systems enabling to switch between interaction modalities in order to select the most suited one for the achievement of the task. This results in an increasing error rate and difficulties to recover from errors [Suhm99].
- *Shortcoming 3. Lack of genuine platform mobility:* most of the current mobile platforms do not allow users to take full advantage of their capabilities as they lack the ability to switch between interaction modalities (e.g., eyes-free, hands-free, audio-only) [Bert05].
- *Shortcoming 4. Lack of usable multimodal UIs:* even if multiple interactions are available, there is a lack of systems that convey information using the modalities that are most appropriate to the end users and their tasks [Rous05].
- *Shortcoming 5. Lack of robust systems:* the traditional GUIs are sometimes less robust than the multimodal systems which benefit from a less complex syntax, higher fluency and doubtless debit [Ovia99].
- *Shortcoming 6. Lack of device effectiveness:* as devices continue to get smaller, the lack of multimodal capabilities decreases the quality of interaction [Ovia99].
- *Shortcoming 7. Lack of multimodal experience:* with the continuously growing number of new devices, there is a lack of experience with the employed multimodal interactions [Hura03]. Therefore, the use of such interactions should be increased by all means possible and encouraged to be accepted widely.
- *Shortcoming 8. Lack of multimodal applications deployment:* although several real MM systems have been built, most of them are:
 - Difficult to generate due to the multitude of devices and their different capabilities and hard to implement as creating a MM UI is more difficult than designing for voice or graphics alone [Aneg04].
 - Too specific to a particular issue.
 - Rarely oriented towards information systems.
 - Often providing solutions to very complex tasks.
 - The result of a manual implementation, which is very specific, non reusable and hard to instal.

Moreover, their number is still reduced compared to the high frequency of existent monomodal applications. Therefore, a high number of users are not aware of the existence of such systems and the benefits they could bring to the HCI.

2.5.3 Requirements

Our methodology, as defined in Section 1.4.1, is delineated by a set of requirements that are elicited and motivated by: on the one hand, the concerns identified in Section 1.2 and on the other hand, by the shortcomings emphasized in Section 2.5.2. which lead us to conclude that the development of MM UIs can be improved along several dimensions. These requirements are defined hereafter in a decreasing order of importance for each dimension of the methodology: (1) Modeling requirements, (2) Method requirements and (3) Tool requirements.

Modeling requirements:

Requirement 1. Support for multimodal input/output: states that our ontology should enable multiple (i.e., at least two different) input/output interaction modalities. The current requirement is motivated by the definition of the multimodal systems (Section 1.3.4).

Requirement 2. Separation of modalities: states that the concepts and the specifications corresponding to each modality should be syntactically separated one from the other. The current requirement is motivated by two aspects: (1) flexibility in the development process given by the possibility to specify separately the UI corresponding to each involved interaction modality and to further combine them altogether, (2) reusability, totally or partially, of the specification corresponding to an interaction modality in other applications that employ it. This requirement contributes to the principle of separation of concerns [Dijk76].

Requirement 3. Support for CARE properties concerning the input/output modalities: states that our ontology should ensure the support of the CARE properties for input/output modalities. This requirement is motivated by the design facilities offered by the CARE properties when defining the relationships that can occur between input/output modalities.

Requirement 4. Ability to model a user interface independent of any modality: states that the provided ontology should ensure a level in the development life cycle that allows to specify a modality-independent UI. This requirement is motivated by the increasing number of novel devices and consequently of interaction modalities that will determine the development of new UIs with new modality capabilities. A modality-independent level will also enable to avoid the redeployment of UIs from scratch. This requirement contributes to the principle of separation of concerns [Dijk76].

Requirement 5. Extendibility to new modalities: states that the ontology structure should allow the extension with new types of interaction modalities. This requirement is motivated by the constant emergence of new computing platforms, each of them supporting a new set of interaction modalities. This requirement is a principle that we

would like to cover, but we are well aware that very complex interactions cannot be supported.

Requirement 6. Ontology homogeneity: states that the ontological concepts should be defined according to a common syntax. The requirement is motivated by the necessity of defining a single formalism for model concepts in order to facilitate their integration and processing.

Requirement 7. Human readability: states that the proposed ontology should be legible by human agents. The current requirement is motivated by two aspects: (1) the need to define in an explicit manner the ontological concepts in order to ensure their precise comprehension, (2) the necessity of sharing the underlying concepts among the research community.

Method requirements:

Requirement 8. Approach based on design space: states that our development life cycle towards a final multimodal UI should be guided by a set of design options. This requirement is motivated by the need to clarify the development process in a structured way in terms of options, thus requiring less design workload.

Requirement 9. Method explicitness: states that the component steps of our methodology should define in a comprehensive way their logic and application. This requirement is motivated by the lack of explicitness of the existing approaches in describing the proposed transformational process.

Requirement 10. Method extendibility: refers to the ability left to the designers to extend the development steps proposed in a methodology. The current requirement is motivated by the lack of flexibility in the current methodological steps that hinders designers to add, delete, modify and reuse these steps.

Tool requirements

Requirement 11. Machine processability of involved models: states that the provided ontology should be proposed in a format that can be legible by a machine. This requirement is motivated by the necessity of transposing the ontological concepts into representations that can be processed by machines.

Requirement 12. Support for tool interoperability: refers to the possibility of reusing the output provided by one tool into another. This requirement is motivated by the lack of explicitness of transformations due to their heterogeneous formats that prevents the reuse of transformations outside the context for which they were designed.

2.6 Conclusion

This chapter presented the existing multimodal frameworks, UIDLs and tools that were considered to bring a significant contribution to the current thesis. The characteristics of a set of languages surveyed in the literature were summed-up and compared in Table 2-1. As a result twelve requirements were elicited that will further argue the thesis statement and validate the results provided by our methodology (Figure 2-1).

3 Conceptual Modeling of Multimodal User Interfaces

3.1 Introduction

After identifying the requirements of MM applications in Chapter 2, the current chapter introduces the concepts of our framework. Section 3.2 presents the selection of the UIDL, whereas Sections 3.3, 3.4 and 3.5 describe the semantics, the syntax and the stylistics of the selected language, respectively.

3.2 Selection of a User Interface Description Language

The objective of the current dissertation is supported by a model-based approach that is intended to offer designers the capability of developing MM UIs of ISs. In software engineering, model-based approaches rely on the power of models to construct and reason about ISs. The goal of these approaches is to propose a set of abstractions, development processes and tools that further enable an engineering approach for UI development. In order to achieve this goal a UIDL is desirable.

3.2.1 Towards choosing a suitable UIDL

For this purpose two solutions were considered: (1) introducing a new specification language or (2) reusing or expanding an already existing UI description language.

Starting from scratch with a specification language requires a lot of efforts before reaching a significant level of interest. Thus, the first solution appears to be time-consuming. With respect to the second solution, we have considered several existing MM languages for which a set of shortcomings have been identified:

- X+V:
 - ✓ Is an implementation language and not a UI Description Language. As such, X+V will be used in the current dissertation as a target language and not as a specification language.
 - ✓ There is no modality-independent level (Requirement 4. Ability to model a UI independent of any modality).
 - ✓ There are no design options in the development life cycle (Requirement 8. Approach based on design space).
- XISL:
 - ✓ There is no modality-independent level (Requirement 4. Ability to model a UI independent of any modality).

3. Conceptual Modeling of Multimodal Web User Interfaces

- ✓ The specification language does not specify the interaction modalities separately (Requirement 2. Separation of modalities).
- ✓ There are no design options in the development life cycle (Requirement 8. Approach based on design space).
- TeresaXML:
 - ✓ Is based on a design space approach but it is limited in terms of alternatives of design options.
 - ✓ The tool is based on a transformational approach, but the transformations are precomputed and hard-coded. Thus, modifiability and extendibility are not supported (Requirement 10. Method extendibility).
 - ✓ As the transformations are hard-coded, they are not expressed in the same language as the specification language (Requirement 6. Ontology homogeneity).

To the above identified shortcomings a more general one is added: whenever we would like to submit an extension of an existing language there is no guarantee that the Consortium in charge with that language will consider it.

3.2.2 UsiXML – the selected UIDL

After identifying the shortcomings for the above MM languages we also considered UsiXML (User Interface eXtensible Markup Language), a UIDL that allows the specification of various types of UIs such as GUIs, VUIs and 3D UIs. This language was selected to support our model-driven approach due to the following motivations:

- UsiXML is structured according to the four basic levels of abstraction (Figure 3-1) defined by the Cameleon reference framework [Calv03]. The framework represents a reference for classifying UIs supporting multiple target platforms and multiple contexts of use and enables to structure the development life cycle into four levels of abstraction: task and concepts, abstract UI (AUI), concrete UI (CUI) and final UI (FUI). The identification of the four levels and their hierarchical organization is built upon their independence with respect to the context in which the FUI is used. Thus, the Task and Concepts level is computational-independent, the AUI level is modality-independent and the CUI level is toolkit-independent.

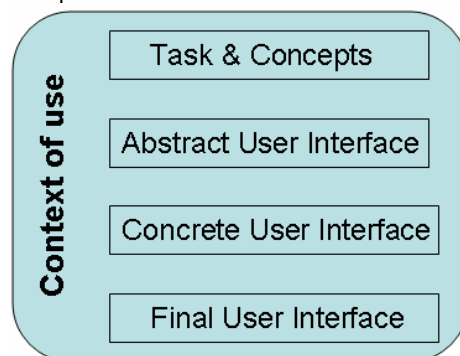


Figure 3-1 Cameleon Reference Framework for multi-target UIs

3. Conceptual Modeling of Multimodal Web User Interfaces

- UsiXML relies on a transformational approach that progressively moves from the Task and Concept level to the FUI
- The steps of the transformational approach define in a comprehensive way their logic and application (*Requirement 9. Method explicitness*).
- The transformational methodology of UsiXML allows the modification of the development sub-steps, thus ensuring various alternatives for the existing sub-steps to be explored and/or expanded with new sub-steps (*Requirement 10. Method extendibility*).
- UsiXML has a unique underlying abstract formalism represented under the form of a graph-based syntax (*Requirement 6. Ontology homogeneity*).
- UsiXML allows reusing parts of previously specified UIs in order to develop new applications. This facility is provided by the underlying XML syntax of UsiXML which allows the exchange of any specification. Moreover, the ability to transform these specifications thanks to a set of transformation rules increases their reusability.
- The progressive development of UsiXML levels is based on a transformational approach represented under the form of a graph-based graphical syntax. This syntax proved to be efficient for specifying transformation rules [Limb04b] and an appropriate formalism for human use (*Requirement 7. Human readability*).
- UsiXML ensures the independence of modality (*Requirement 4. Ability to model a UI independent of any modality*) thanks to the AUI level which enables the specification of UIs that remains independent of any interaction modality such as graphical, vocal or 3D interaction
- UsiXML supports the incorporation of new interaction modalities thanks to the modularity of the framework where each model is defined independently and to the structured character of the models ensured by the underlying graph formalism (*Requirement 5. Extendibility to new modalities*).
- UsiXML is supported by a collection of tools that allow processing its format (*Requirement 11. Machine processability of involved models*)
- UsiXML allows cross-toolkit development of interactive application thanks to its common UI description format (*Requirement 12. Support for toolkit interoperability*).

3.3 Conceptual contribution

The current section emphasises our ontological contribution defined according to UsiXML v1.8 [USIX07] which integrates the improvements and the expansions accomplished by the present thesis in order to adapt the UsiXML models to the requirements of MM UIs. For each model a discussion of its suitability with respect to our MM interaction goals is carried out and solutions are offered whenever shortcomings of the existing ontology defined according to UsiXML v1.6.3 [USIX05] are identified. For the semantics of our ontology UML class diagrams are employed .

3. Conceptual Modeling of Multimodal Web User Interfaces

3.3.1 Task Model

The existing Task Model defined in [USIX05] is an extended version of ConcurTaskTree notation defined in [Pate97]. Due to the consideration of MM UIs, we expanded the existing Task Model in order to better respond to the requirements imposed by these applications (Section 2.5.3). A complete description of the expanded Task Model can be found in Appendix A.

3.3.1.a Existing Task Model

The *Task Model* describes the interactive tasks as viewed by the end user while interacting with the system. It is composed of *tasks* and *task relationships* (Figure 3-2). *Tasks* are, notably, described with attributes such as *name* and *type*. The *name* of the task is generally expressed as a combination of a verb and a substantive (e.g., consult patient file). The *type* attribute identifies one of the four basic task types: user, interactive, system or abstract.

Leaf tasks are described by two additional attributes (i.e., *userAction* and *taskItem*) that enable a refined expression of the task nature. This expression is based on the taxonomy introduced by [Cons03] that allows qualifying a UI in terms of the abstract actions it supports. The taxonomy is twofold: a verb describes the type of activity at hand and an expression designates the type of object on which the action is operated. By combining these two dimensions a derivation of interaction objects that are supposed to support a task becomes possible.

The *userAction* attribute refers to verbs that indicate the actions required to perform the task (Table 3-1), while the *taskItem* attribute refers to an object type or subject of an action (Table 3-2). The existing values were identified based on Constantine's taxonomy.

userAction	Definition
start	Specifies that an action is triggered
stop	Specifies that an action is ended
select	Specifies a selection between multiple items
create	Specifies the creation of an item
delete	Specifies the deletion of an item
modify	Specifies the modification of an item
move	Specifies the movement of an item
duplicate	Specifies the duplication of an item
toggle	Specifies the toggle between different items
view	Specifies that an item is shown to the user

Table 3-1 Definition of existing values for the *userAction* attribute

taskItem	Definition
element	Specifies that the item has a single characteristic
container	Specifies that the item is an aggregation of elements
operation	Specifies that the item is a function
collection	Specifies that the item is composed multiple elements

Table 3-2 Definitions of existing values for the *taskItem* attribute

3. Conceptual Modeling of Multimodal Web User Interfaces

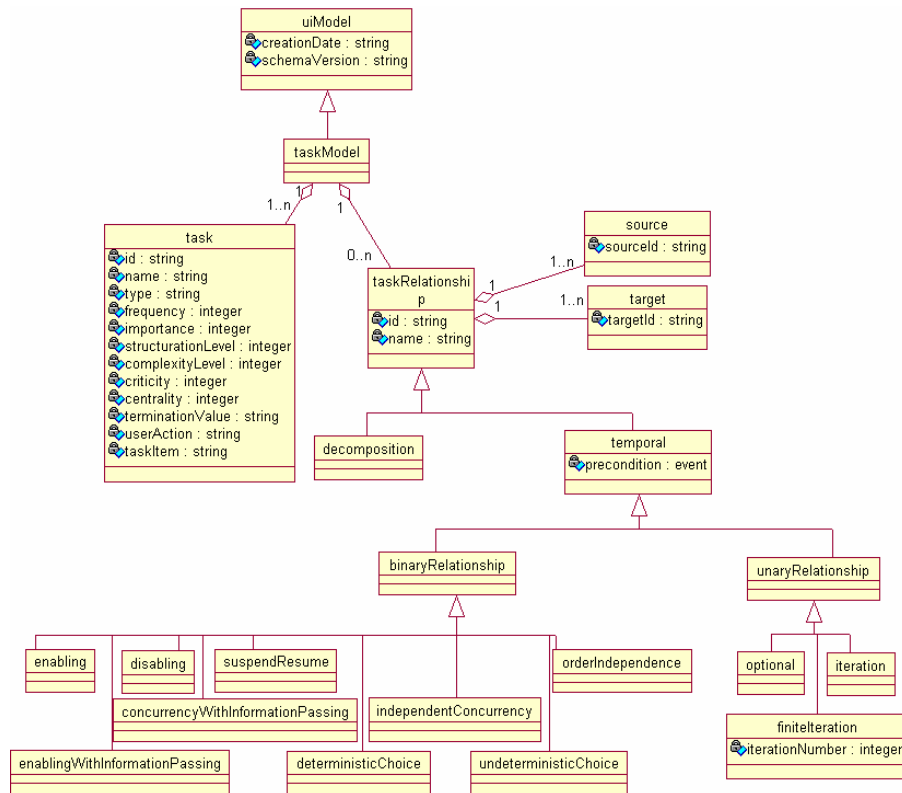


Figure 3-2 Meta-model of the Task Model

3.3.1.b Expanded Task Model for Multimodal User Interfaces

The *userAction* attribute refers to verbs that identify actions from the user point of view. Taking into account the values identified in Section 3.3.1.a, this definition is not generally true. For instance, the *view* value specifies an action from the system’s perspective (i.e., the system displays an item). Consequently, we replaced the name of the attribute with *taskType*, a name that remains independent of the entity that accomplishes the task (i.e., the user or the system). In order to avoid the confusion with the attribute *type*, the latter was renamed *category* while keeping the previously specified semantics.

Moreover, we have added/modified several values of this attribute (Table 3-3). Thus, the *view* value suggests the idea of visualisation of items, while the Task Model should remain modality-independent. Consequently, we replaced it with *convey*, a value which doesn’t make any reference to the employed modality. In addition, the *delete* value specifies that an item is removed, but there is no value specifying that an item is reinitialized (e.g., setting to blank the values of a text filed widget). For this purpose the *erase* value was added. Other taskType values that address the requirements of 3D UIs are introduced and defined in [Gonz06].

taskType	Definition
convey	The item is conveyed to the user
erase	The value of an item is reinitialized

3. Conceptual Modeling of Multimodal Web User Interfaces

Table 3-3 Definition of newly identified values for the *taskType* attributes

The developer should consider the values of the *taskType* attribute identified in Table 3-1 and Table 3-3 only if the Task Model is employed for further reification processes towards the generation of more concrete UI models. Otherwise, the values are not mandatory. Table 3-4 provides a set of possible synonyms that can be used in parallel with the existing ones.

taskType	Synonyms
start	go/to/initiate
stop	end/exit/finish/complete
select	choose
create	input/encode/enter
delete	Eliminate
erase	Efface
modify	change/alter/transform
move	relocate
duplicate	clone/twin/reproduce
toggle	switch
convey	communicate/transmit

Table 3-4 Synonyms for the *taskType* values

The collection value of the *taskItem* attribute identified in Table 3-2 specifies that an item is composed of multiple elements, while a collection could be composed of a series of containers as well. For instance, one container specifying the list of books of an author and another specifying some features for each book (i.e., title, publisher and price) can be grouped into the same collection. Therefore, we split the *collection* value into *collection of elements* and *collection of containers* (Table 3-5).

taskItem	Definition
collection of elements	Specifies that the item is composed of a list of elements
collection of containers	Specifies that an item is composed of a list of containers

Table 3-5 Definitions of newly identified values for the *taskItem* attribute

By combining the *taskType* and *taskItem* attributes a series of possible situations could occur (Table 3-6).

taskType	taskItem	Example
start	operation	Start to look for the definition of a word in an online dictionary
select	element	Select the gender of a person
create	element	Input an email address in a form
	element	Convey the result of a computational operation (the result can be expressed graphically by displaying it on the screen or vocally by system utterance)
	container	Convey the starting date of a conference (the day,

3. Conceptual Modeling of Multimodal Web User Interfaces

convey		month and year can be displayed on the screen or can be uttered by the system)
	collection of elements	Convey the list of authors of a book (the list of authors can be displayed or can be uttered by the system)
	collection of containers	Convey the list of books of an author, by specifying for each one the title, the editor, the publisher, the price, etc. (each feature can be displayed or can be uttered by the system)

Table 3-6 Examples of combinations between values of `taskType` and `taskItem` attributes

3.3.2 Domain Model

The *Domain Model* described in [USIX05] did not benefit from any conceptual contribution as its specification is suitable for the requirements imposed by the development of MM UIs. This model is a description of the classes of objects manipulated by a user while interacting with the system (Figure 3-3). It consists of one or more *domainClasses*, and potentially one or more *domainRelationships* between these classes.

A *class* describes the characteristics of a set of objects sharing a set of common properties. The concepts identified at the class level are: *attributes*, *methods* and *objects*. An *attribute* is a particular characteristic of a class that is described by several features: *attributeDataType* refers to basic data types as string, integer, real, boolean or enumerated (*enumerated* describes an attribute that has a value from a set of enumerated items). The *attributeCardMin* and *attributeCardMax* describes, respectively, the lower and upper bound of the attribute cardinality (0 for a not mandatory attribute and 1 for a mandatory one). A *method* is the description of a process able to change the system's state and is described by its signature (i.e., name and input and output parameter(s)). An *object* is an instance of a class composed of attribute instances that are able to call methods.

A *domainRelationship* describes various types of relationships between classes and can have three types: generalization, aggregation or ad hoc. Class relationships are described thanks to several attributes that enable to specify their role names and cardinalities.

3. Conceptual Modeling of Multimodal Web User Interfaces

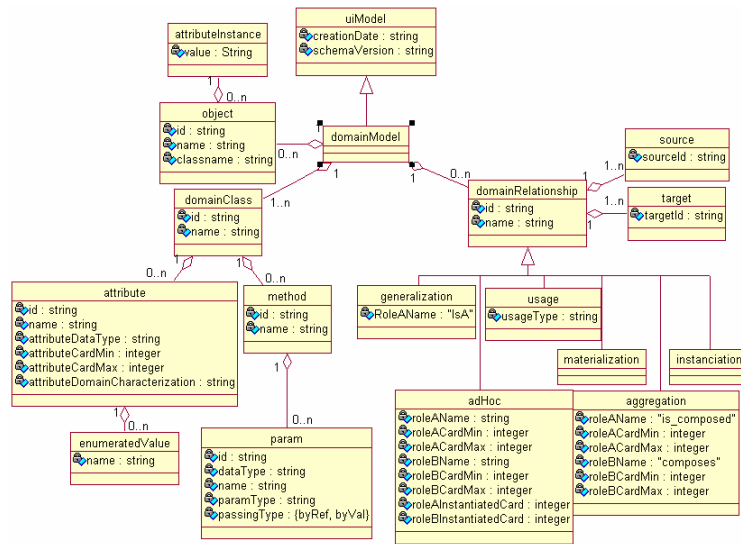


Figure 3-3 Meta-model of the Domain Model

3.3.3 Abstract User Interface Model

The *Abstract User Interface Model* described in [USIX05] did not benefit from any conceptual contribution as its specification is suitable for the requirements imposed by the development of multimodal UIs. This model represents a canonical expression of the renderings and manipulations of the domain concepts and functions in a way that is independent of any interaction modality and computing platform. Therefore, there is no information regarding the manner in which this abstract specification will be concretized: graphical, vocal or multimodal. This concretization is achieved in the next level.

The AUI Model (Figure 3-4) is populated with *Abstract Interaction Objects (AIOs)* between which *Abstract User Interface Relationships* have been defined.

3. Conceptual Modeling of Multimodal Web User Interfaces

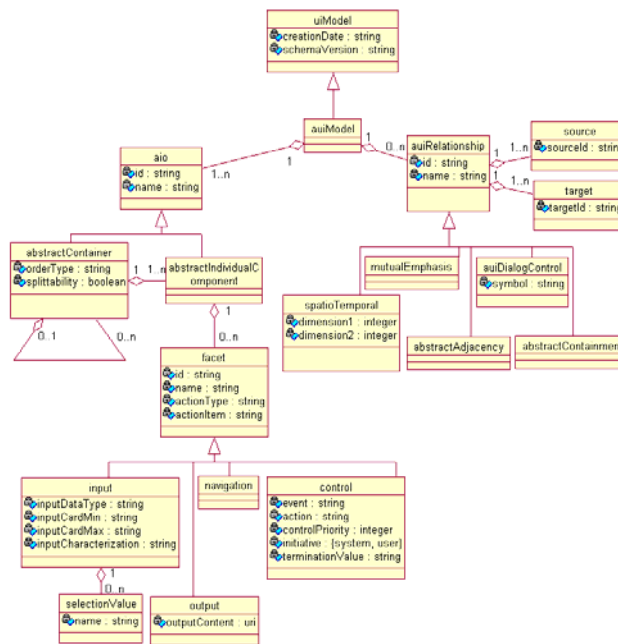


Figure 3-4 Meta-model of the AUI Model

AIOs are abstraction of widgets found in most of the popular graphical toolkits (e.g., windows, buttons) and vocal toolkits (e.g., prompts, vocal menus). They can have two types: *Abstract Individual Components (AICs)* or *Abstract Containers (ACs)*.

An *AIC* is any individual element populating an *AC*. An *AIC* assumes at least one basic system interaction function described as a *facet* in the UI. As *AICs* are composed of multiple facets, we call them multi-faceted. Each facet describes a particular function an *AIC* may assume. We identify four main facets:

1. *Input facet*: specifies that an input information is accepted by the *AIC*.
2. *Output facet*: specifies that an output data is conveyed to the user by the *AIC*.
3. *Navigation facet*: specifies that the *AIC* enables a container transition.
4. *Control facet*: specifies that the *AIC* enables to trigger methods from the Domain Model.

An *AIC* may assume several facets simultaneously. For instance, an *AIC* may display an output while accepting an input from a user or trigger a container transition and a method defined in the Domain Model.

The *actionType* attribute of a facet enables the specification of the type of action an *AIC* allows to perform. The *actionItem* attribute characterizes the item manipulated by the *AIC*. As the *AUI Model* and the *Task Model* are both modality-independent, the values of *actionType* and *actionItem* of the former model can be inherited (Figure 3-5) from the *taskType* and *taskItem* attributes defined in the *Task Model*, respectively.

3. Conceptual Modeling of Multimodal Web User Interfaces

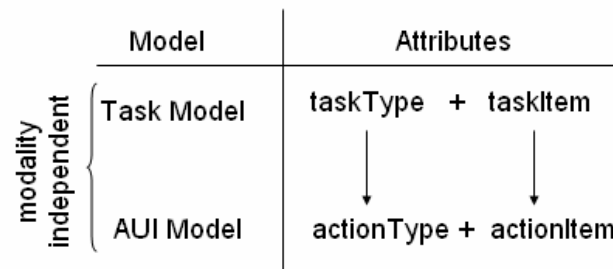


Figure 3-5 Abstract attribute values inheriting Task attribute values

Even if there is no conceptual contribution brought to this level of specification, an identification of the different types of structures an instruction can have and their specification according to the AUI Model has been made. This will help us determine the components of an instruction and the possible cardinalities of their instances at a level that is still modality-independent. In addition, each of these components could be further concretized in a particular modality or combination of modalities giving rise to a MM instruction.

We started our research taking into account the general structure of an instruction in ISs. According to [IBM93] natural languages typically have significantly more nouns than verbs, and a graphical UI typically contains more objects than actions. Just as the same verb can be applied to many nouns, the same action can be applied to many objects, independent of the type of UI, be it graphical, vocal, MM, etc. Therefore, the action/object paradigm is defined as a pattern for interaction in which a user selects an action and an object to apply it to. But objects are usually endowed with features which help us characterize them. Therefore, in ISs these features were transposed into parameters assigned to objects. Consequently, the general structure of an instruction is composed of three elements (Figure 3-6) that could have single or multiple cardinality or even could be optional depending on the context in which the instruction is used:

$$Instruction := \{Action, Object, Parameter\}$$

Figure 3-6 The general structure of an instruction in ISs

The models composing our ontology support this structure as follows: the *actionType* attribute identifies the type of action(s) the instruction applies, the *actionItem* specifies the type of object(s) on which the action is applied, whereas the *parameter(s)* describing the object(s) are feature(s) stored in the Domain Model.

Based on these observations, four types of instructions have been identified. For each type we provide two MM examples: (1) consists of vocal fulfillment of form-based UI and (2) allows users to interact vocally and graphically in order to manipulate different objects on a map. It is worth noticing that whenever multiple *actionTypes* are applied on the same *actionItem* the considered *object* can be identified:

- *Directly*: by re-specifying it for each *actionType* (E.g., “Create a *blue lake*. Select the *blue lake* and move the *blue lake* under the green park”).

3. Conceptual Modeling of Multimodal Web User Interfaces

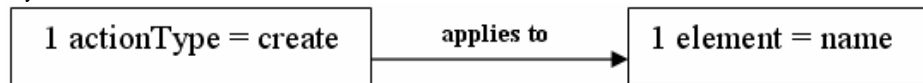
- *Indirectly*: by using deictic words in order to avoid multiple occurrences of the same *object*. For instance, words like *this*, *that*, *it*, *there*, *here* can be employed to substitute objects which have been previously introduced (e.g., “Create a blue lake. Select *this* (pointing gesture towards the blue lake) and move *it* under the green park”).

1. 1 *actionType* applied to 1 *actionItem* (*element*)

Example 1 (form-based UI):

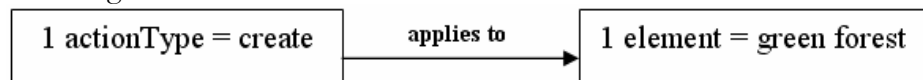
C: “What is your name?”

U: “My name is Peter.”



Example 2 (direct manipulation UI):

U: “Create a green forest.”

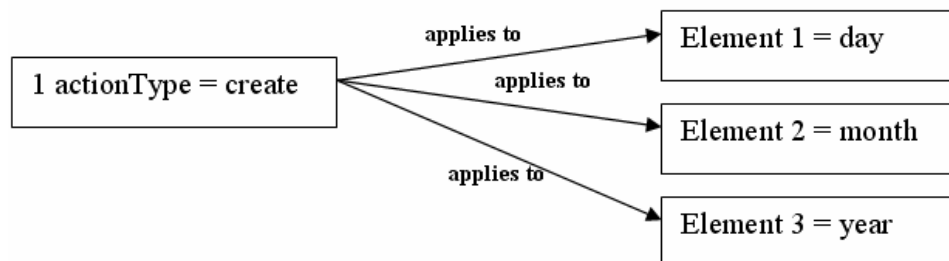


2. 1 *actionType* applied to *N* *actionItems* (*elements*)

Example 1 (form-based UI):

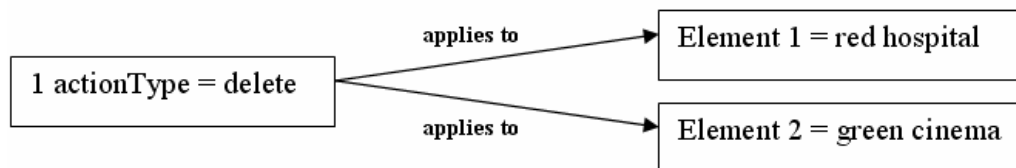
C: “Please specify your birthday.”

U: “My birthday is on 8th of February 1986.”



Example 2 (direct manipulation UI):

U: “Delete the red hospital and the green cinema.”



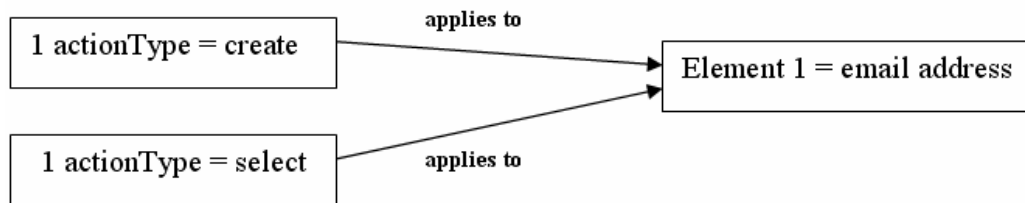
3. *N* *actionTypes* applied to 1 *actionItem* (*element*)

Example 1 (form-based UI):

C: “Please say your email address.”

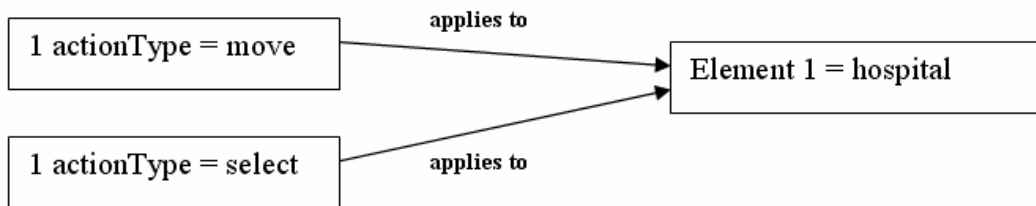
U: “My email is johnson@yahoo.com. Select this e-mail.”

3. Conceptual Modeling of Multimodal Web User Interfaces



Example 2 (direct manipulation UI):

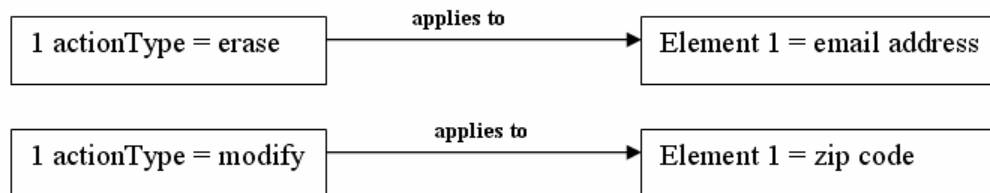
U: "Move the hospital next to the police office and select it."



4. N actionTypes applied to N actionItems (elements)

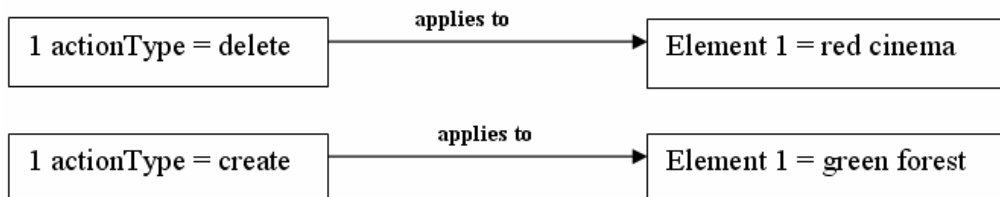
Example 1 (form-based UI):

U: "Erase the email address and modify the zip code to 1020."



Example 2 (direct manipulation UI):

U: "Delete the red cinema and create a green forest in its place."



AUI Relationships are abstract relationships among AUI objects. Relationships may have multiple sources and multiple targets. There are a couple of types of relationships, among which:

- *AbstractAdjacency*: allows to specify an adjacency constraint between two AIOs
- *AbstractContainment*: allows to specify that an AC embeds one or more ACs or one or more AICs
- *AuiDialogControl*: enables the specification of the dialog control in terms of LOTOS operators between AIOs.

3. Conceptual Modeling of Multimodal Web User Interfaces

3.3.4 Concrete User Interface Model

The *Concrete User Interface Model* described in [USIX05] benefit from a conceptual contribution as its specification was not suitable for the requirements imposed by the development of MM UIs. The benefits consist mainly in improved and expanded definitions of the vocal UI description.

This model allows both the specification of the presentation and the behavior of an UI with elements that can be perceived by the users [Limb04b]. The CUI abstracts a FUI in a definition that is independent of programming toolkit peculiarities.

CUI Model (Figure 3-7) concretizes the AUI for a given context of use into *Concrete Interaction Objects/Components (CIOs/Components)* and *Concrete User Interface Relationships* so as to define layout and/or interface navigation of 2D graphical and/or vocal widgets.

CIOs realize an abstraction of widget sets found in popular graphical and vocal toolkits (e.g., Java AWT/Swing, HTML 4.0, Flash DRK 6, VoiceXML). A CIO is defined as an entity (e.g., window, push button, text field, check box, vocal output, vocal input, vocal menu) that can be perceived and/or manipulated by the users. Due to the graphical and vocal consideration of UsiXML, CIOs are further divided into: *graphicalCIOs* and *vocalCIOs*. Details regarding the types of *graphicalCIOs*, *vocalCIOs* and *Concrete User Interface Relationships* are provided in the following section.

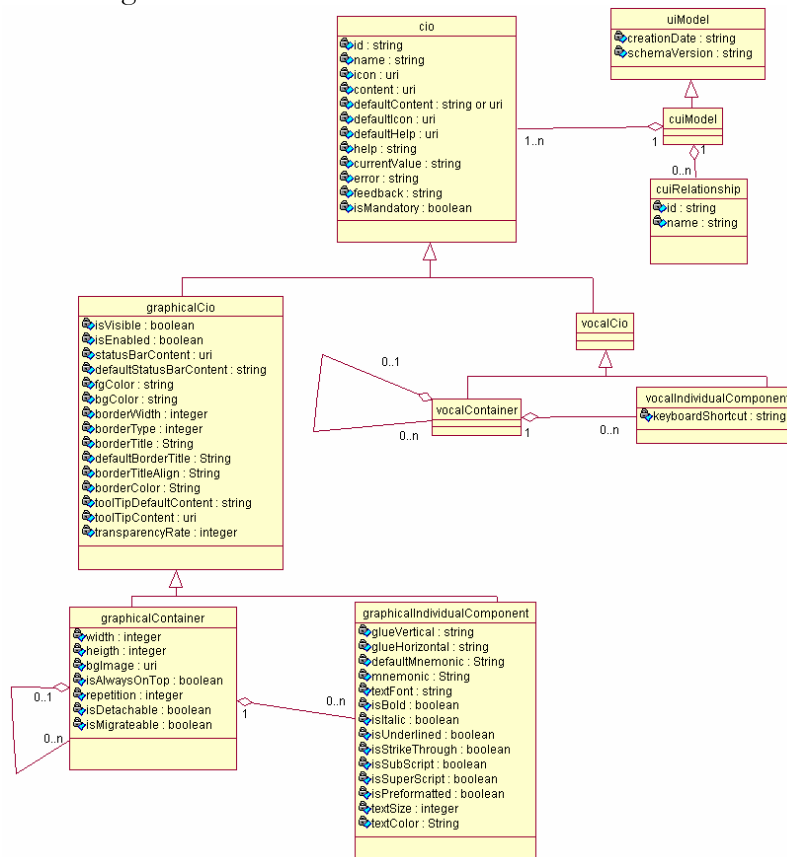


Figure 3-7 Excerpt of the CUI Meta-model

3. Conceptual Modeling of Multimodal Web User Interfaces

3.3.5 Mapping Model

The *Mapping Model* described in [USIX05] did not benefit from any conceptual contribution as its specification is suitable for the requirements imposed by the development of MM UIs. This model contains a series of related mappings between models or elements of the models (Figure 3-8). A Mapping Model serves to gather a set of pre-defined, inter-model relationships that are semantically related. It consists of one to more *interModelRelationships*, a part of them being used throughout the steps of the transformational approach:

- *Manipulates*: maps a task onto a domain concept (i.e., a class, an attribute, a method or any combination of these types).
- *Updates*: is a mapping between any AUI or CUI component and a domain attribute or run time instantiated attribute. It enables to specify that a UI component provides a value for the related domain concept.
- *Triggers*: indicates a connection between a method of the Domain Model and a AUI or CUI individual component.
- *IsExecutedIn*: indicates that a task is performed through one or several ACs and AICs.
- *IsReifiedBy*: maps the elements of the AUI onto elements of the CUI. This relationship specifies the manner in which any AIO can be reified by a CIO.

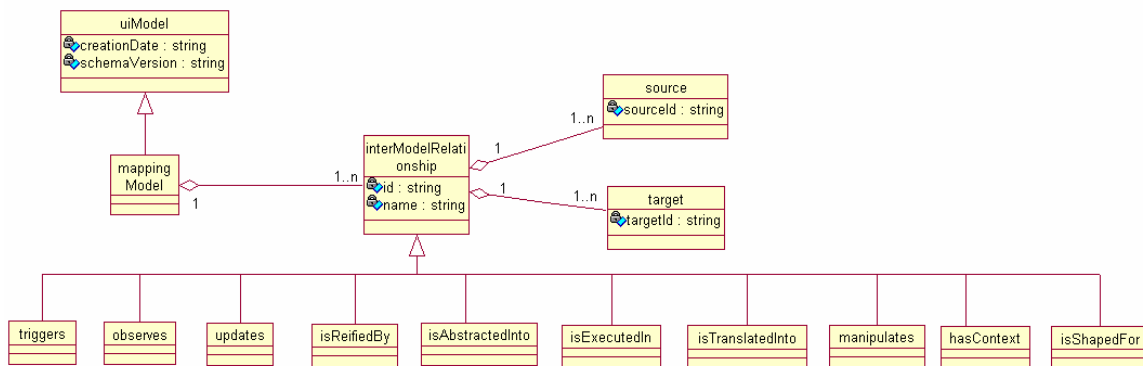


Figure 3-8 Meta-model of the Mapping Model

3.3.6 Transformation Model

The *Transformation Model* described in [USIX05] did not benefit from any conceptual contribution as its specification is suitable for the requirements imposed by the development of MM UIs.

This model (Figure 3-9) is conceptualizing rules that enable the transformation of a model specification (at a certain level of abstraction) into another or adapting this specification for a new context of use. A transformation rule realizes a unit transformation operation on a model and is composed of:

- *LHS (Left Hand Side)*: models the pattern that will be matched in the host model.
- *RHS (Right Hand Side)*: models the part that will replace the LHS in the host model.
- *NAC (Negative Application Condition)*: models the condition that has to hold false before trying to match LHS into the host model.

3. Conceptual Modeling of Multimodal Web User Interfaces

- *AttributeCondition*: is a textual expression indicating a condition scoping on element attributes of the lhs of a transformation rule.
- *RuleMapping*: defines the source and the target models of the transformation rule. For instance, a rule may establish a mapping between a *Task Model* and an *Abstract Model*. In this case, the source indicates the source model of the mapping, while the target indicates the target model.

Transformation rules are applied in order to develop UIs following a specific development path (e.g., forward engineering, reverse engineering, adaptation to context of use). A development path is composed of development steps that can imply three types of transformations depending on the development direction:

- *Reification*: consists in the derivation of the next lower model in our reference framework
- *Abstraction*: consists in the derivation of the next upper model in our reference framework
- *Translation*: is a type of model transformation adapting a set of UI models to a target context of use.

A development step is decomposed into development sub-steps. A development sub-step is always realized by a single transformation system. A transformation system is composed of a set of sequentially applied transformation rules. One transformation system applies one sub-derivation unit [Limb04]. A sub-derivation unit is defined as a collection of derivation rules that realize a basic development activity. A basic development activity has been identified to sub-goals assumed by the developer while constructing a system (e.g., choosing widgets, defining navigation structure).

3. Conceptual Modeling of Multimodal Web User Interfaces

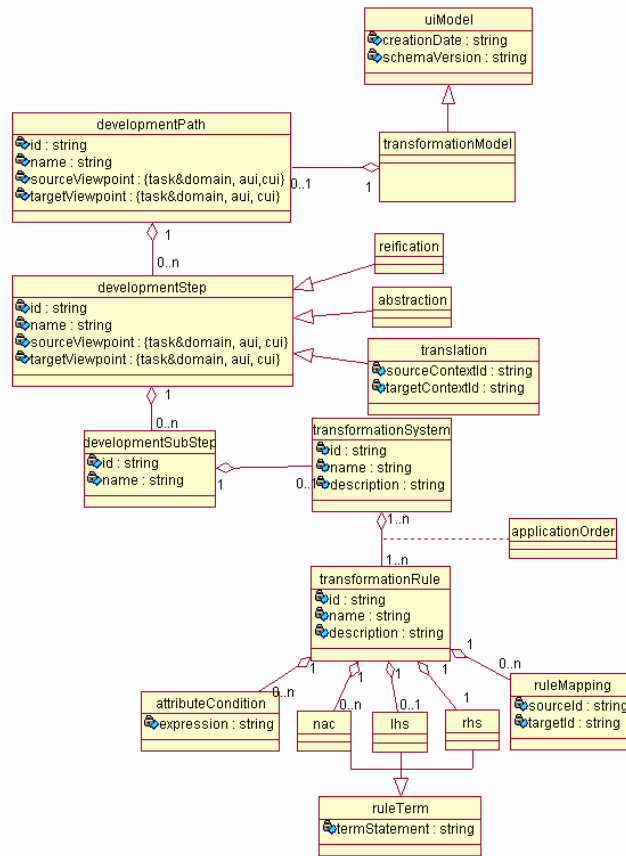


Figure 3-9 Meta-model of the Transformation Model

3.4 Semantics of the multimodal interaction objects

Semantics (in Latin letters *semantikós*, or *significant meaning*, derived from *sema*, translated as *sign*) is the study of meaning, in some sense of a term. Hereafter we provide the semantics of the CIOs composing the CUI Model and of the relationships defined between them.

3.4.1 Semantics of the Graphical Concrete Interaction Objects

No modifications were brought to the semantics of the *Graphical CIOs* described according to [USIX05]. These objects are divided into Containers and Individual Components. *Graphical Containers (GCs)* (Figure 3-10) contain a collection of CIOs (either GICs or GCs) that support the execution of a set of logically/semantically connected tasks. Hereafter we define the semantics of a couple of containers used in the current thesis:

- *Window*: is a container that can be found in almost all 2D graphical toolkits. A window may contain other GCs.
- *Box*: is a container that enables an unambiguous structuring of GICs within a window, a *tabbedItem*, a *dialogBox*. Boxes are embedded one into the other. Their type may be: *main* (i.e., the topmost box in a container), *horizontal* or *vertical*.

3. Conceptual Modeling of Multimodal Web User Interfaces

- *GroupBox*: allows to group a set of GICs. A group of option buttons is a typical use of a groupBox. Normally a groupBox does not contain any other GC.
- *TabbedDialogBox*: is a group of dialogBoxes where each dialogBox is accessible via a tab mechanism. A tabbed dialogBox is composed of tabbedItems.
- *ToolBar*: is a bar containing a series of selectable buttons that give the user an easy way to select different items.
- *MenuPopUp*: is a menu of commands or options displayed when an item is selected. The selected item is generally at the top of the display screen and the menu is displayed just below it.

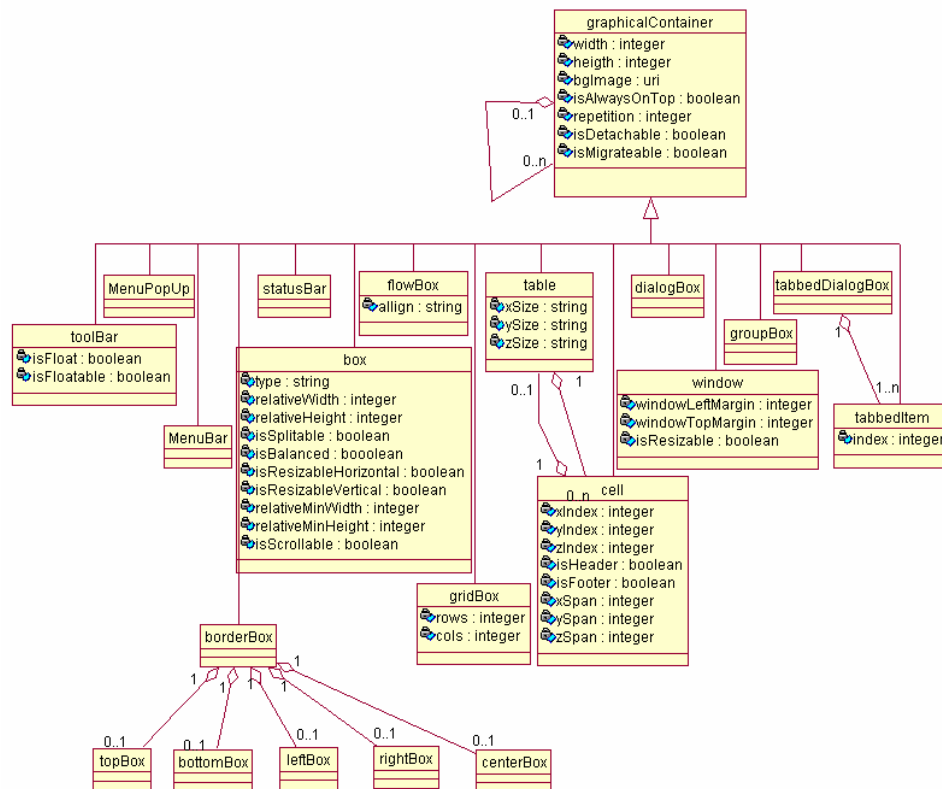


Figure 3-10 Graphical containers

Graphical Individual Components (GICs) are objects contained in GCs. Figure 3-11 illustrates a part of GICs defined in [USIX05] for which we offer the semantics:

- *InputText*: is a GIC specialized in handling input textual content.
- *OutputText*: is a GIC specialized in handling output textual content.
- *Button*: is alternatively called trigger button as it aims to trigger any kind of action available in the system.
- *Checkbox*: enables a boolean choice by checking a square box aside of a label.
- *RadioButton*: enables a boolean choice by checking a circle aside of a label. A group of optionButtons differentiates from a group of checkBoxes by its mutual exclusive selection feature.

3. Conceptual Modeling of Multimodal Web User Interfaces

- *ComboBox*: enables a direct selection over a collection of sequentially, predefined items. It might also enable editing new items.
- *ImageComponent*: is a GIC specialized in handling image content.

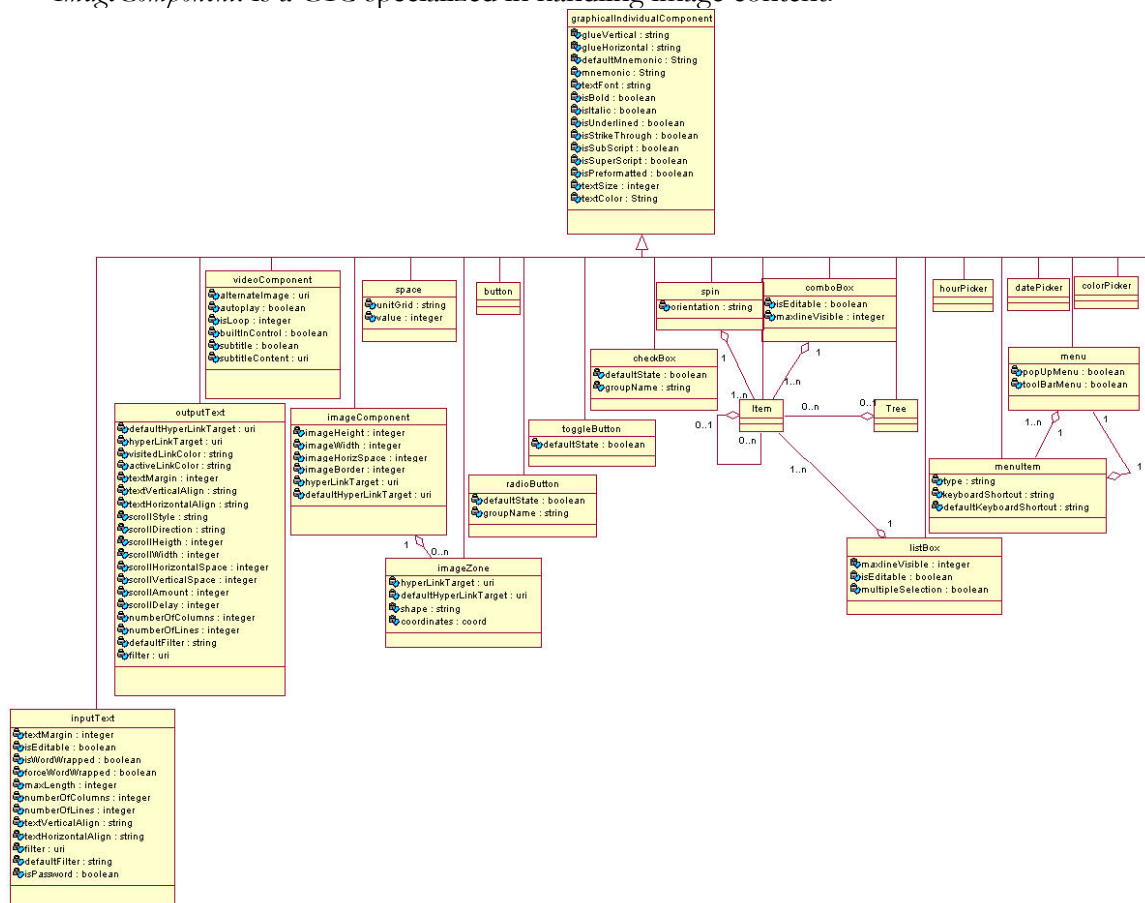


Figure 3-11 Several Graphical Individual Components

3.4.2 Semantics of the Vocal Concrete Interaction Objects

3.4.2.a Existing semantics of the Vocal Concrete Interaction Objects

The existing vocal ontology described in [USIX05] consists of concepts that support the vocal interaction thanks to *Auditory Interaction Objects* and *Auditory relationships*. The former can be *Auditory Containers* representing a logical grouping of other containers or *Auditory Individual Components*. These individual concepts can have two types: *auditoryOutput* supporting music, voice or a simple earcon (i.e., an auditory icon) or *auditoryInput* which is a mere time slot allowing users to provide an auditory input using their voice or any other physical device able to produce sound. Auditory relationships can have two types:

- *AuditoryTransition*: enable to specify a transition between two auditory containers.
- *AuditoryAdjacency*: indicates the time adjacency between two auditory components.

By observing the current ontology described in [USIX05] we were able to identify that the semantics of the vocal concrete interaction objects suffers from a set of shortcomings:

3. Conceptual Modeling of Multimodal Web User Interfaces

- It does not provide a specialized container that enables a dialog between the system and the end-user (i.e., synthesize/collect data from the system/user). This might prove to be useful in order to better distinguish between containers that support a user/system interaction and those that act just as basic containers used for grouping purposes.
- It does not provide a specialized vocal container that allows users to choose between different options. This is extremely useful as a traditional vocal dialog often consists of multiple choice questions.
- It does not allow to identify the elements of an instruction: the utterances identified in the grammar specify the tasks as a whole without mapping them with a corresponding part (i.e., action, object). As a consequence, the grammar content cannot be reused.
- It does not allow to define the element's order of utterance: there are no means to specify an alternative between two or more utterances, a sequence of utterances or a particular order of utterances. Consequently, all possible combinations between the elements of an instruction have to be explicitly specified in the grammar. This will result in a high number of possible combinations that will further increase with the growing of the number of elements.
- It does not allow to specify the visibility of the grammar: grammars could be made visible only in the current vocal forms or to other forms in the current document.
- It does not allow to specify the cardinality of an element utterance: for instance one user would like to utter "Select the ship" while some others: "Select and delete the ship". It can be observed that in the first utterance there is only one action defined (i.e., "select"), whereas in the second one there are two actions (i.e., "select and delete").
- It does not allow to specify the language in which the utterances have to be pronounced in order to be recognized by the system (e.g., English, French)
- It does not specify explicitly the type of the system's output. The output could provide the user with synthesized prompt information or with some feedback following a previously processed input.
- It does not allow to play audio pre-recorded files or to record user's vocal messages
- It does not allow to interrupt the execution of the current container or of the entire application. For instance the end-user would like to put end to a dialog which does not provide any useful information or to stop interacting with the application due to an unexpected outer system task.

3.4.2.b Expanded semantics of the Vocal Concrete Interaction Objects

Based on the shortcomings identified above, [USIX07] expands the existing vocal ontology offering a larger set of vocalCIOs (Figure 3-12) that cover the requirements of vocal and MM UIs (Requirement 1. Support for multimodal input/output). By analogy with the graphicalCIOs, the vocalCIOs are divided into Containers and Individual Components. *Vocal Containers (VCs)* represents a logical grouping of other VCs or VICs and inherit the *isOrderIndependent* attribute which indicates if the inputs of the container can be filled in any order or not:

3. Conceptual Modeling of Multimodal Web User Interfaces

- *VocalGroup*: is the root element of all vocalCIOs. Acts as a basic container for all VCs and VICs.
- *VocalForm*: enables a dialog whose purpose is to synthesize/collect data from the system/user.
- *VocalMenu*: allows to choose among different vocalMenuItems. The *currentValue* attribute is employed to store user's input.
- *VocalConfirmation*: requests from the user a confirmation of a previous input. It is composed of a *vocalPrompt* that solicits the confirmation followed by a *vocalInput* gathering the user's input. For instance, "Do you want to delete this file? Say Yes or No."

VocalIndividualComponents (VICs) are vocalCIOs contained in a VCs. All VICs inherit the attribute *keyboardShortcut* that is the DTMF representation of the output, where the possible values are {0-9, #, *}. The following VICs were introduced:

- *VocalOutput*: is an object used to synthesize data to the user. This data is specified in the attribute *defaultContent* inherited from the CIO class. The *volume* attribute specifies the sound volume expressed in Db (decibel). The *intonation* attribute expresses the dominant tone according to which the vocalOutput will be synthesized: positive, negative, interrogative, exclamative. *Pitch* is the perceptual attribute of a vocalOutput which enables the user to locate the sound on a scale from low (1) to high (5). An attribute *isInterruptible* specifies if the vocalPrompt can be interrupted by a user's utterance. A vocalOutput can be further sub-divided into:
 - *VocalFeedback*: provides users with some feedback following a previously processed vocalInput. For example: "Your answer was: male".
 - *VocalPrompt*: provides users with prompt information that will be synthesized. If there is an audio file to be played, the attribute *audioSource* specifies its URI.
 - *VocalMenuItem*: specifies a menu item belonging to a *vocalMenu*. The DTMF sequence corresponding to this item is specified by the *dtmf* attribute. For example: the sequence of strokes 1-3-5 will select directly this vocal item. The *attached* attribute specifies the reference to the next document (an external reference expressed as an uri) or to the next *vocalContainer* in the current document (its *id* expressed as a string) attached to this item.
 - *Audio*: is employed to play audio prerecorded files. The *audioSource* attribute specifies the URI of the audio file to be played or the name of the reference where the recorded file is stored. The *errorMessage* attribute indicated the synthesized error message to be played by the system if the audio file is not technically available.
- *VocalInput*: is an object used to gather input from the user by speech recognition or audio recording. The *elapsedTime* attribute is the time frame expressed in seconds during which the user is allowed to utter the input. The recognized input is stored in the *currentValue* attribute. The *defaultContent* attribute replaces the use of a grammar for the following values:

3. Conceptual Modeling of Multimodal Web User Interfaces

- *Boolean*: used for *Yes* and *No* answers. For DTMF inputs, *1* stands for affirmative and *2* for negative.
- *Date*: used for input that specifies a date (i.e., four digits for the year, two digits for the month and two digits for the day) are allowed.
- *Digits*: used for input that specifies digits from 0 through 9.
- *Currency*: used for input that specifies amounts (the format may include a decimal point) and the used currency. The format is: currency name, amount, eventually followed by an amount after the decimal point (e.g., euros fifty point twenty).
- *Number*: used for input that specifies numbers (e.g., one hundred fifty-four).
- *Phone*: used for input that specifies a phone number.
- *Time*: used for input that specifies a time (i.e., the hours and the minutes). The format is: hour, minute followed by AM or PM (e.g., nine twenty five AM)
- *Grammar*: is an structured and compacted enumeration of a set of utterances (i.e., words and phrases) that constitute the acceptable user input for a given vocalInput. The grammar can be internal (i.e., it is specified within the document) or external (i.e., it is specified in an external file whos URI is specified by the *defaultContent* attribute). The *version* attribute indicates which version of the grammar specification is being used (the current version is 1.0). The *language* attribute indicates according to which language the utterance has to be pronounced in order to be recognized by the system. The specification of the language takes the form of the couple: the name of the language followed by the country in which it is used (e.g.: English-UK). The *mainPart* attribute is the first part of the grammar that will be treated by the system. The *mode* attribute specifies the available interaction type. The default type is *voice* for voice-based interaction, whereas for phone-based interaction the value is *dtmf*. The *visibility* attribute specifies the visibility of the grammar. If set to *document* the grammar is active throughout the current document. If set to *form* (the default value) the grammar is active throughout the current vocalForm.
- *Part*: contains other part elements or available input items. The *structure* attribute specifies how the user's inputs should be uttered in order to be recognized by the system. There are three possible values: *choice* (i.e., the grammar items are alternative inputs), *sequential* (i.e., sequence of grammar items that have to be uttered one after another in the order of their appearance) or *asynchronous* (i.e., sequence of grammar items in which the items do not have any particular order of utterance). The *visibility* attribute specifies the visibility of the part component. If set to *private* (the default value) the part component can be used only by the containing grammar. If set to *public* the part component can be referenced by other grammars. The *multiplicity* attribute indicates how many times the enclosed items may be repeated. The default value is 1. The multiplicity is defined as follows:
 - X (*where* $X > 0$): the items are repeated exactly X times.
 - X - Y (*where* $0 \leq X < Y$): the items are repeated between X and Y times (inclusive).
 - X - (*where* $X \geq 0$): the items are repeated X or more times.The *language* attribute indicates in which language the items have to be pronounced in order to be recognized by the system. The specification of the language takes the form

3. Conceptual Modeling of Multimodal Web User Interfaces

of the couple: the name of the language followed by the country in which it is used (e.g., French-CA). If it is not specified, it inherits the value from the language attribute of the embedding grammar element.

- *Item*: enables to specify a grammar input or to reference another part element. The grammar input is specified by the *defaultContent* attribute. The same attribute is used to specify the referenced part as a string containing the “#” symbol followed by the name of the part element. The *language* attribute indicates in which language the item has to be pronounced in order to be recognized by the system. The specification of the language takes the form of the couple: the name of the language followed by the country in which it is used (e.g.: French-CA). The attribute allows to mix multiple languages in the same grammar. If it is not specified, it inherits the value from the language attribute of the embedding part element.
- *VocalNavigation*: ensures the dialog transfer between vocal CIOs. The *nextContainer* attribute transfers the dialog to a VC embedded either in the current document or in another document. The value of the attribute is composed of # followed by the *id* of the VC. The *nextComponent* attribute transfers the dialog to another VIC in the current VC. The value of the attribute is composed of # followed by the *id* of the VIC. The *evalContainer* attribute evaluates an ECMA Script expression that yields the document to which the dialog will be transferred. If the expression is evaluated to TRUE, the first choice is considered, while if it is FALSE the second choice is considered. The *evalComponent* attribute evaluates an ECMA Script expression that yields the VIC from the current VC to which the dialog will be transferred. If the expression is evaluated to TRUE, the first choice is considered, while if it is FALSE the second choice is considered. The *isBridgeable* attribute indicates if the source document remains active during the navigation.
- *Connect*: enables to connect a grammar element to a dialog transition. The *nextContainer* attribute transfers the dialog to a VC embedded either in the current document or in another document. The value of the attribute is composed of # followed by the *id* of the vocalGroup. The *evalContainer* attribute contains an ECMA Script expression that is evaluated to determine the name of the VC to which the dialog is transferred. The VC is embedded either in the current document or in another document.
- *Record*: is an object used to record a vocal message of the user. The *defaultContent* attribute contains the URI of the recorded audio file or the name of the reference to this file that can be further played using the *audio* element. If the *beep* attribute is set to TRUE, an acoustic beep is emitted by the system announcing the availability of the recording. If set to false (the default value) no beep is emitted and the user can start to record immediately after the prompt. The *elapsedTime* attribute specifies the maximum time period during which the user is allowed to record the message. It is expressed in milliseconds or seconds (e.g., "100ms" or "2s"). The *silenceTime* attribute is the silence time period that determines the record to be stopped. It is expressed in milliseconds or seconds. If the *dtmfEnabled* attribute is set to TRUE (i.e., the default value), it enables the users to press a key in order to stop the recording.

3. Conceptual Modeling of Multimodal Web User Interfaces

- *Submit*: is employed in order to send data to the server and/or to ensure the dialog transfer between vocal CIOs. The *defaultContent* attribute specifies the URI of the file towards which the information should be send. The *expr* attribute specifies an ECMA script expression that is evaluated to determine dynamically the URI of the reference file. The *varList* attribute contains the list of variables to submit. When the list is not specified, all the variables of the *vocalInputs* in the current *vocalForm* are submitted. When specified, the list may contain individual variable names of *vocalInputs* and/or declared variables. The *audioFetch* attribute contains the URI of the audio clip to play while the submit element is being processed. The *timeoutFetch* attribute specifies the interval to wait for the content to be returned before throwing an error event. This interval can be expressed in milliseconds or seconds. The *nextContainer* attribute transfers the dialog to a VC embedded either in the current document or in another document. The value of the attribute is composed of # followed by the *id* of the VC.
- *vocalVar*: used to declare a variable. The *defaultContent* attribute contains the name of the variable that will hold the result. The *currentValue* attribute specifies the initial value of the variable. If no initial value is provided, the variable will hold the value *undefined*.
- *setVar*: used to set a previously declared variable to a specific value. The *defaultContent* attribute specifies the name of the variable to set, while the *currentValue* attribute indicates the new value of the variable.
- *resetVar*: used to clear a previously declared variable. The *defaultContent* attribute specifies the list of variables to be reset. When it is not specified, all variables in the current *vocalForm* are reset.
- *If*: it conditions the execution of certain parts of the document. The *guard* attribute is a condition that has to hold true in order to execute the instructions coming after the *if* element.
- *Else*: is an optional element embedded in the *if* element. It allows executing the instructions coming after it if the guard condition did not hold true.
- *Elseif*: optional element embedded in the *if* element. It is used to test more then two possible results. The *guard* attribute is a condition that has to hold true in order to execute the instructions coming after the *if* element.
- *Break*: interrupts the execution of the current VC.
- *Exit*: terminates the execution of the vocal application.

There are four possible values of event types that can be associated to vocalCIOs. These values are specified by the *eventType* attribute of the *event* element:

- *Error*: catches all events of type error.
- *Help*: catches a help event.
- *NoInput*: catches a no input event.
- *NoMatch*: catches a no match event.

3. Conceptual Modeling of Multimodal Web User Interfaces

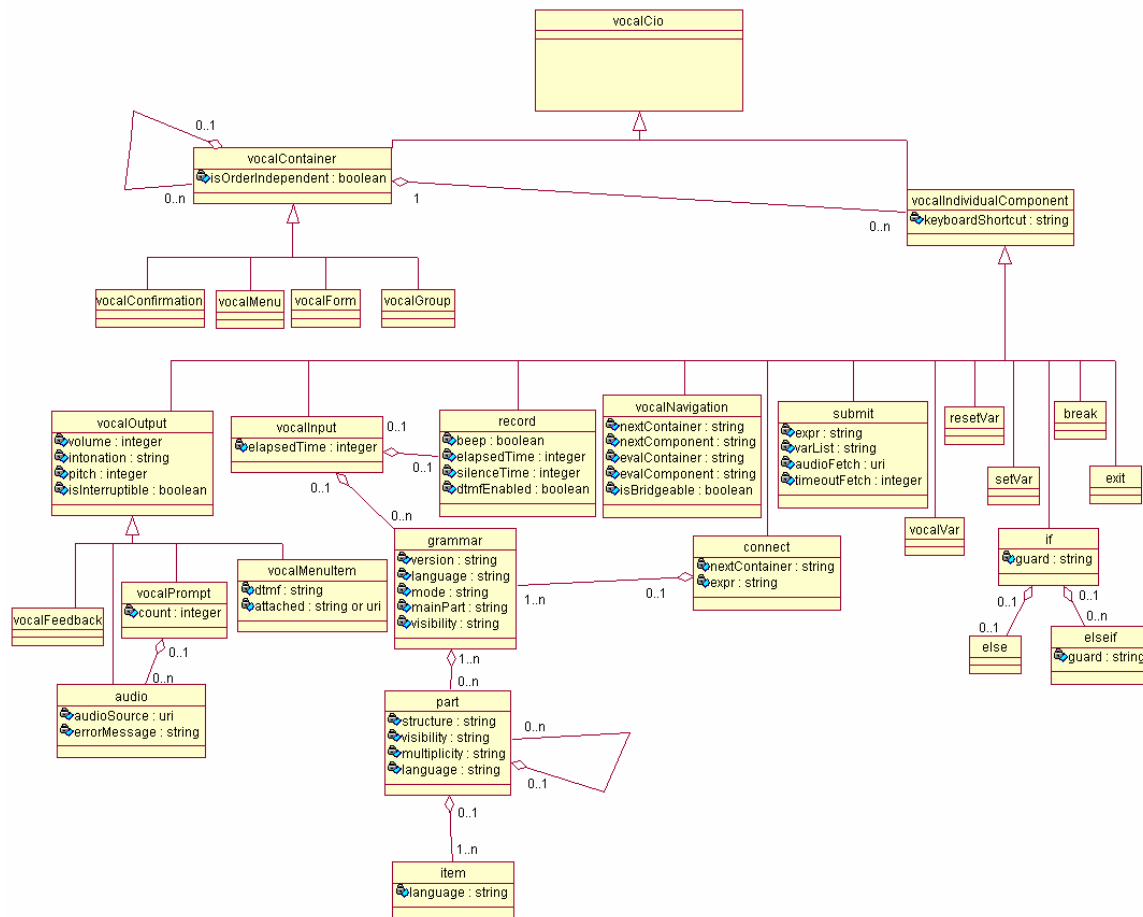


Figure 3-12 Vocal Concrete Interaction Objects

For a better understanding of the concepts defined above we exemplify graphically two vocal interactions between the system (S) and the user (U). The first dialog (Figure 3-13) describes the fulfillment of the *Provide age task* by an end-user. The involved vocal CIOs are described in the order of dialog flow:

- *VocalGroup*: is the upper most VC that contains all vocalCIOs involved in the dialog.
- *VocalForm*: is the VC that contains all the vocalCIOs involved in the dialog.
- *VocalPrompt*: is the VIC employed to invite the user to input the age.
- *VocalInput*: is the VIC that gathers the user’s input (the age) by speech recognition with the *defaultContent* attribute set to *number*.
- *VocalConfirmation*: is the VC which requires the confirmation of the recognized input.
- *VocalFeedback*: is the VIC that provides the user with the feedback regarding the previously recognized input.
- *VocalPrompt*: is the VIC inviting the user to confirm the previously provided feedback.
- *VocalInput*: is the VIC that gathers the user’s confirmation by speech recognition with the *defaultContent* attribute set to *boolean*.

3. Conceptual Modeling of Multimodal Web User Interfaces

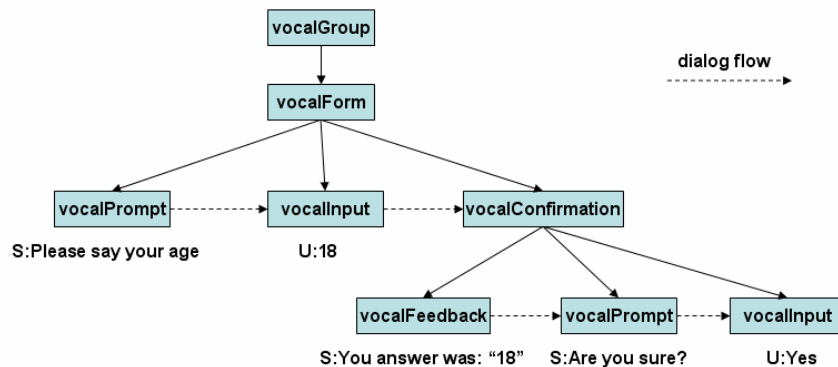


Figure 3-13 VocalCIOs involved in the fulfillment of Provide age task

The second dialog (Figure 3-14) describes a vocal application of a phone line company where users can select among different options. It consists of two sub-tasks: first the user provides the name to the system and then selects among three proposed options in a menu. The involved vocalCIOs are described in the order of dialog flow:

- *VocalGroup*: is the upper most VC that contains all vocalCIOs involved in the dialog.
- *VocalForm*: is the VC containing the vocalCIOs involved in the fulfillment of the first sub-task.
- *VocalPrompt*: is the VIC used to welcome the user to the vocal application of the phone line company and invites to input the name.
- *Record*: is the VIC that records the user's input (i.e., the name). The name of the reference to the recorderd file is stored in the *defaultContent* attribute.
- *VocalMenu*: is the VC that allows to select among different options. The selected option recognized by the system is stored in the *currentValue* attribute.
- *VocalMenuItem1*: is the VIC used to modify the personal info. For this purpose it is connected to a vocalForm specified by the *attached* attribute
- *VocalMenuItem2*: is the VIC used to select the move-out line option. For this purpose it is connected to a vocalForm specified by the *attached* attribute.
- *VocalMenuItem3*: is the VIC used to require bill info. For this purpose it is connected to a vocalForm specified by the *attached* attribute.
- *VocalFeedback*: is the VIC that provides the user with the feedback regarding the recognized input. Based on this input the dialog continues with the corresponding vocalForm.

3. Conceptual Modeling of Multimodal Web User Interfaces

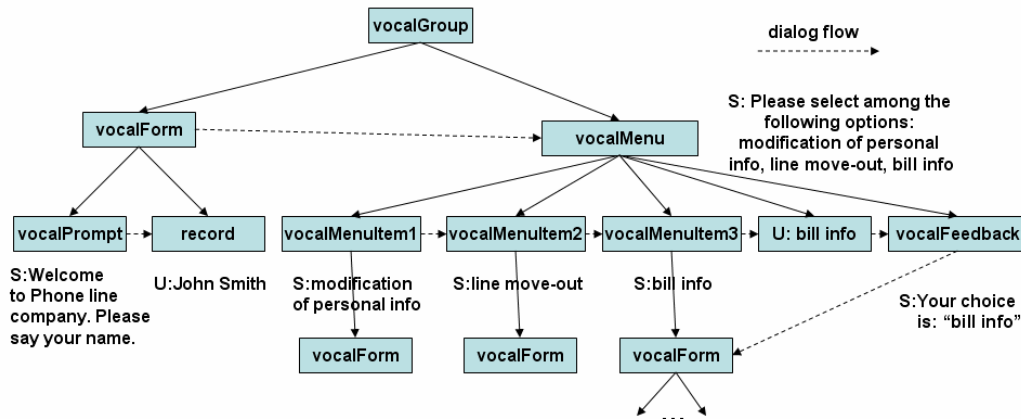


Figure 3-14 VocalCIOs used for a vocal application of a Phone line company

3.4.3 Semantics of the Multimodal Concrete Interaction Objects

MultimodalCIOs are obtained by combining *graphicalCIOs* and *vocalCIOs*. For a set of popular widgets, Table 3-7 identifies a possible correspondence with the proposed CIOs for three types of interactions: graphical, vocal and MM. A correspondent rendering for each one of them is illustrated as well. For graphical and MM UIs, we consider the *imageComponent* element consisting of representative icons that enable to guide the user with available types of interaction or to specify the type of vocal feedback provided by the system:

1. Label:

- **G:** ensured by *outputText*.
- **V:** ensured by *vocalPrompt*.
E.g.: System (*vocalPrompt*): “Welcome to the UCL web site”
- **MM:** ensured by *outputText*, *vocalPrompt* and *imageComponent* (loud speaker icon).
E.g.:
System displays the welcome message (*outputText*): Welcome to the UCL web site.
System welcomes the user vocally (*vocalPrompt*): “Welcome to the UCL site”.

2. Label + Text field:

- **G:** ensured by *outputText*, *inputText* and *imageComponent* (keyboard icon).
- **V:** ensured by *vocalPrompt* and *vocalInput*.
E.g.
System (*vocalPrompt*): “Please say your name”.
User’s input (*vocalInput*) is recorded in a file (*record*): “John Smith”.
- **MM:** ensured by *outputText*, *inputText*, *vocalPrompt*, *record*, *audio* and *imageComponent* (microphone and keyboard icons to specify the available input interactions and loud speaker icon to indicate the vocal feedback).
E.g.:
User clicks on the *Name* label (*outputText*).
System (*vocalPrompt*): “Please say your name”.
User’s input (*vocalInput*) is recorded in a file (*record*): “John Smith”.
System displays the recorded input (*inputText*): John Smith.

3. Conceptual Modeling of Multimodal Web User Interfaces

System plays the recorded file (*audio*): “Your name is John Smith”.

3. Label + Combo box:

- **G:** ensured by *outputText*, *comboBox* with *items* and *imageComponent* (mouse and keyboard icons).
- **V:** ensured by *vocalPrompt*, *vocalInput* and *grammar* with *items*.

E.g.:

System (*vocalPrompt*): “Select the credit card type. Choose between Visa, MasterCard or American Express”.

User selects among the different proposed credit card types (*grammar* with *items*) the desired one (*vocalInput*): “Visa”.

- **MM:** ensured by *outputText*, *comboBox* with *items*, *vocalPrompt*, *vocalInput*, *grammar* with *items*, *vocalFeedback* and *imageComponent* (microphone and keyboard icons to specify the available input interactions and loud speaker icon to indicate the vocal feedback).

E.g.:

User clicks on the *Credit Card* label (*outputText*).

System invites the user to choose between different credit cards (*vocalPrompt*): “Select the credit card type. Choose between Visa, MasterCard and American Express.”

User selects among the different proposed credit card types (*grammar* with *items*) the desired one (*vocalInput*): “Visa”.

System displays the recognized input (*comboBox*): Visa.

System (*vocalFeedback*): “Your choice is: Visa.”.

4. Group of radio buttons:

- **G:** ensured by a *groupBox* embedding a set of *radioButtons* and *imageComponent* (mouse icon).
- **V:** ensured by *vocalPrompt*, *vocalInput* and *grammar* with *items*.

E.g.:

System (*vocalPrompt*): “Please say your gender. Choose between male and female”.

User selects among the different options (*grammar* with *items*) the gender (*vocalInput*): “Male”.

- **MM:** ensured by a *groupBox* embedding a set of *radioButtons*, *vocalPrompt*, *vocalInput*, *grammar* with *items* and *imageComponent* (microphone and mouse icons to specify the available input interactions).

E.g.:

User clicks on the *Gender* label (*groupBox label*).

System invites the user to select the gender (*vocalPrompt*): “Please say your gender. Choose between male and female”.

User selects among the different options (*grammar* with *items*) the gender (*vocalInput*): “Male”.

System displays the recognized input by checking the corresponding item (*radioButton*): male.

5. Group of check boxes:

- **G:** ensured by a *groupBox* embedding a set of *checkboxes* and *imageComponent* (mouse icon).
- **V:** ensured by *vocalPrompt*, *vocalInput* and *grammar* with *items*.

3. Conceptual Modeling of Multimodal Web User Interfaces

E.g.:

System (*vocalPrompt*): “Please select your hobbies. Choose among the following options: sports, travels, music, movies”.

User selects among the different options options (*grammar* with *items*) the preferred hobbies (*vocalInput*): “Sport and music”.

- **MM:** ensured by a *groupBox* embedding a set of *checkboxes*, *vocalPrompt*, *vocalInput*, *grammar* with *items* and *imageComponent* (microphone and keyboard icons to specify the available input interactions).

E.g.:

User clicks on the *Hobbies* label (*groupBox label*).

System invites the user to select the hobbies (*vocalPrompt*): “Please select your hobbies. Choose among the following options: sports, travels, music, movies”.

User selects among the different options options (*grammar* with *items*) the preferred hobbies (*vocalInput*): “Sport and music”.

System displays the recognized input by checking the corresponding items (*checkboxes*): sports and music.

6. Label + List box:

- **G:** ensured by an *outputText*, *listBox* with *items* and *imageComponent* (mouse icon).
- **V:** *vocalPrompt*, *vocalInput* and *grammar* with *items*.

E.g.:

System (*vocalPrompt*): “Please choose your favorite singers: Chris Hay, Lee Hardy, Paul Sheerin,...”.

User select among the different options (*grammar* with *items*) the favourite singer (*vocalInput*): “Lee Hardy”.

- **MM:** ensured by *outputText*, *listBox* with *items*, *vocalPrompt*, *vocalInput*, *grammar* with *items* and *imageComponent* (microphone and mouse icons to specify the available input interactions).

E.g.:









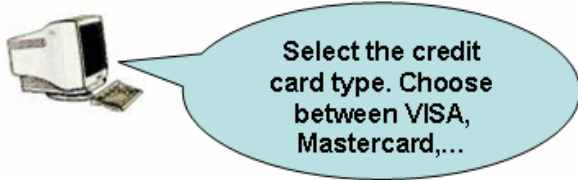



User clicks on the *Singers* label (*outputText*).

System invites the user to select the singers (*vocalPrompt*): “Please choose your favorite singer: Chris Hay, Lee Hardy, Paul Sheerin,...”.

User select among the different options (*grammar* with *items*) the favourite singer (*vocalInput*): “Lee Hardy”.

System displays the recognized input by selecting the corresponding item (*listbox item*): Lee Hardy.

3. Conceptual Modeling of Multimodal Web User Interfaces

Widgets	User Interface type		
	Graphical interaction	Vocal interaction	Multimodal (graphical and vocal) interaction
1. Label	outputText Welcome to the UCL web site	vocalPrompt 	outputText + vocalPrompt + imageComponent Welcome to the UCL web site 
2. Label + Text field	outputText + inputText + imageComponent Name  <input type="text"/>	vocalPrompt + vocalInput + record 	outputText + inputText + vocalPrompt + record + audio + imageComponent Name   <input type="text"/> 
3. Label + Combo Box	outputText + comboBox + items Card type  <div style="border: 1px solid gray; padding: 2px;"> VISA VISA MASTERCARD AMERICAN EXPRESS </div>	vocalPrompt + vocalInput + grammar + items 	outputText + comboBox + items + vocalPrompt + vocalInput + grammar + items + vocalFeedback + imageComponent Card type   <div style="border: 1px solid gray; padding: 2px;"> VISA VISA MASTERCARD AMERICAN EXPRESS </div> 
4. Group of radio buttons	groupBox + radioButtons	vocalPrompt + vocalInput + grammar + items	groupBox + radioButtons + vocalPrompt + vocalInput + grammar + items +

3. Conceptual Modeling of Multimodal Web User Interfaces

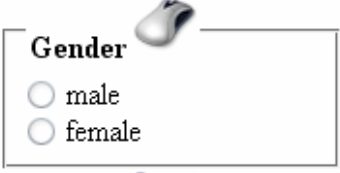

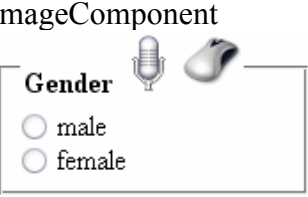

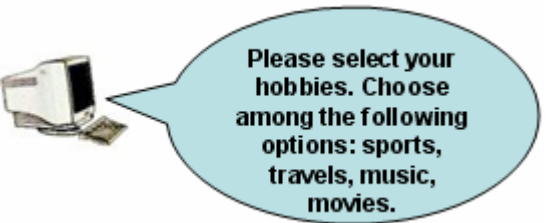




			<p>imageComponent</p> 
<p>5. Group of check boxes</p>	<p>groupBox + checkBoxes + imageComponent</p> 	<p>vocalPrompt + vocalInput + grammar + items</p> 	<p>groupBox + checkBoxes + vocalPrompt + vocalInput + grammar + items + imageComponent</p> 
<p>6. Label + List Box</p>	<p>outputText + listBox + items + imageComponent</p> 	<p>vocalPrompt + vocalInput + grammar + items</p> 	<p>outputText + listBox + items + vocalPrompt + vocalInput + grammar + items + imageComponent</p> 

Table 3-7 Correspondence between popular widgets and CIOs of different modalities

3.4.4 Semantics of the Concrete User Interface Relationships

3.4.4.a Existing semantics of the Concrete User Interface relationships

The *Concrete User Interface Relationships* described in [USIX05] map two or more CIOs. These relationships always have at least one source object and at least one target object. There are three types of relationships:

1. *GraphicalRelationship*: maps two or more graphicalCIOs and is sub-divided in:
 - *GraphicalTransition*: maps one or several GCs by specifying a navigation structure among the different containers populating a CUI Model. The *transitionType* attribute identifies the following values: *open*, *close*, *minimize*, *maximize*, *suspend/resume*. The *transitionEffect* attribute defines the animation type to be used when a graphical transition is ensured from a source container to a target container (e.g., wipe, box in, box out, fade in, fade out, dissolve, split).
 - *GraphicalAdjacency*: enables to specify an adjacency constraint between two graphicalCIOs. An adjacency relationship is inferred from the order in which components are placed into horizontal and vertical boxes. Consequently, it is never explicitly stated in the specification.
 - *GraphicalContainment*: enables to specify that a GC embeds one or more GCs or one or more GICs. The relationship is particularly useful for adding or deleting GICs from a GC.
 - *GraphicalAlignment*: specifies an alignment constraint between two GICs.
 - *GraphicalEmphasis*: enables to specify that two or more GICs are differentiated in some way (e.g., with different color attributes).
2. *VocalRelationship*: maps two or more vocalCIOs and is sub-divided in:
 - *VocalTransition*: enables to specify a transition between two VCs. The *transitionType* attribute determines the type of transition (e.g., open, mute, reduce volume, restore volume). By analogy with the graphical counterpart relationship we extend the existing set of values with two new ones: *activate* and *deactivate*. The *transitionEffect* attribute allows a specification of an auditory effect to the transition (e.g.: fade-out, fade-in).
 - *VocalAdjacency*: enables to specify an adjacency constraint between two vocal CIOs. The *delayTime* attribute expresses a delay in milliseconds between two vocal elements.
 - *VocalContainment*: allows to specify that a VC embeds one or more VCs/VICs. This relationship is particularly useful for adding or deleting VICs from VCs.
3. *CuiDialogControl*: enables the specification of the dialog control in terms of LOTOS operators between any types of CIOs, be it graphical, vocal or combined. In the current thesis we adopt this specification, but some other techniques, such as the notation proposed in [Winc08], could be considered. In MM UIs, one has to give a special attention to the dialog control between elements [IBM03b]. For instance, if the voice control moves from a text field to a list box, the designer should make sure that the visual focus is also moved from the text field to the list box. Conversely, if the visual focus initiates the transition, the voice should respond accordingly.

3.4.4.b Expanded semantics of the Concrete User Interface Relationships

By observing the existing ontology described in [USIX05] we were able to identify that the semantics of the CUI relationships suffers from a set of shortcomings. Therefore an expansion illustrated in Figure 3-15 Concrete UI Relationships is provided according to [USIX07]:

- When navigating between two sub-tasks the designer typically considers two simultaneous actions that seem to appear natural during the HCI: deactivate the GC in which the source sub-task is executed and activate the GC in which the target sub-task will be executed. However, these actions are not explicitly specified by [USIX05]. Therefore, in order to offer a more precise identification of transition types between GCs we extend the existing set of values by introducing two new ones: *activate* and *deactivate*.
- The existing ontology did not allowed to specify the synchronization between the graphical and the vocal components. Synchronization is an issue specific to the MM environments. Since the current work considers MM applications using both vocal and graphical interactions, they should always be synchronized [IBM03b]. Therefore, we introduce hereafter the *synchronization* relationship which synchronizes the information manipulated by the vocalCIOs and graphicalCIOs in a MM UIs. The two types of interaction objects specified in our ontology are syntactically separated one from the other (Requirement 2. Separation of modalities). The synchronization relationship ensures that:

- Vocal input is returned to both vocalCIOs and graphicalCIOs
- Graphical input updates both vocalCIOs and graphicalCIOs.

For instance, if the user has to fill in his/her name in a *textField* (i.e., a GIC) by employing the vocal modality (i.e., a *vocalInput* is employed), the recognized result is updating the values in both *currentValue* attributes of the VIC and of the GIC. In addition, if the user is typing the name, the introduced value is updating the values in both *currentValue* attributes of the VIC and of the GIC.

Four cases when the synchronization relationship were identified:

- *Synchronization between 1 VIC and 1 GIC*: is defined directly between the VIC (i.e., the source) and the GIC (i.e., the target). For instance, if we consider the task of vocally filling in the name of a person in a text field, the designer has to ensure the synchronization between the *vocalInput* and the *textField*.
- *Synchronization between 1 VIC and n GICs*: is defined between the VIC (i.e., the source) and the GC (i.e., the target) that embeds the n GICs. For instance, we consider the task of vocally filling in the date in a form by using three combo boxes (one for the day, one for the month and one for the year). If the designer desires to enable the fulfillment of the task all at once (e.g., “5th of May 2006”) then he/she has to embed all three *combo boxes* in the same GC (e.g., a *groupBox*) that will be synchronized with the *vocalInput* recognizing the user’s input.
- *Synchronization between n VIC and 1 GIC*: the synchronization will be defined between the VC (i.e., the source) that embeds the VICs and the GIC (i.e., the target). For instance, if we consider the task of vocally selecting the date in a date picker widget by using three separate *vocalInput* objects (one for the day, one for

3. Conceptual Modeling of Multimodal User Interfaces

the month and one for the year), the designer has to embed all three *vocalInput* objects in the same VC (e.g., *vocalForm*) that will be synchronized with the GIC.

- *Synchronization between n VIC and n GIC*: implies a decomposition process in order to reach one of the three situations described above. If the designer wants to reach the first identified situation where 1 VIC is synchronized with 1 GIC, then the source and the target of the synchronization relationship will be established by the order in which the VICs and GICs appear.

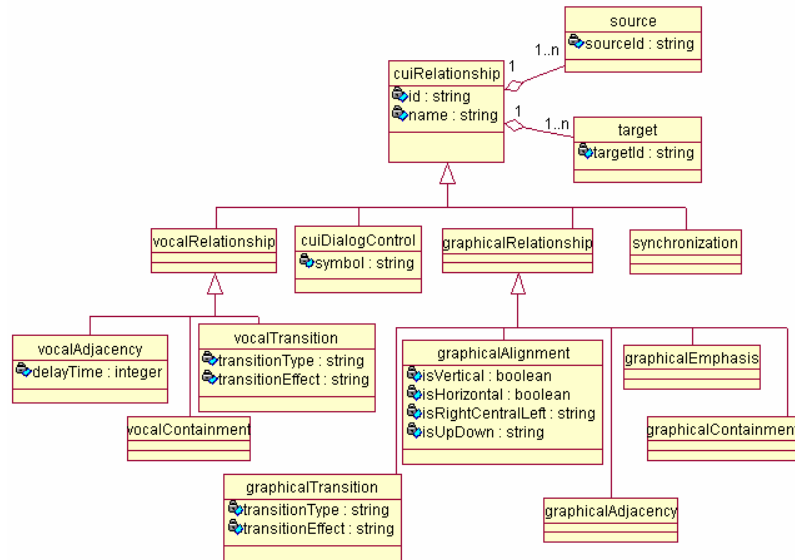


Figure 3-15 Concrete UI Relationships

3.5 Syntax of UsiXML

As motivated in Section 3.2.2, the selected UIDL to support our ontology is the UsiXML language. This section specifies its syntax as a support of the semantics of the ontology introduced in Section 3.4. Syntax is often opposed to semantics: while the latter pertains to what something means, the former pertains to the formal structure in which something is expressed.

3.5.1 From Semantics to Concrete Syntax

On the one hand, the semantics of our ontology is defined by employing UML class diagrams. On the other hand, the syntax of the UsiXML language has an XML-based format structure which allows to describe sets of data with a tree-like structure. Figure 3-16 illustrates how the ontological concepts defined in the previous section are transformed in a UsiXML specification which considers XML Schemas [W3C01] for the definition of valid XML elements. For this purpose manual transformations (T1) are applied in order to produce UsiXML XML Schemas from the UML class diagram description. Objects resulting from the instantiations of class diagram concepts are further transformed (T2) into UsiXML specification. Finally, the UsiXML specification is validated by the corresponding XML Schema.

3. Conceptual Modeling of Multimodal User Interfaces

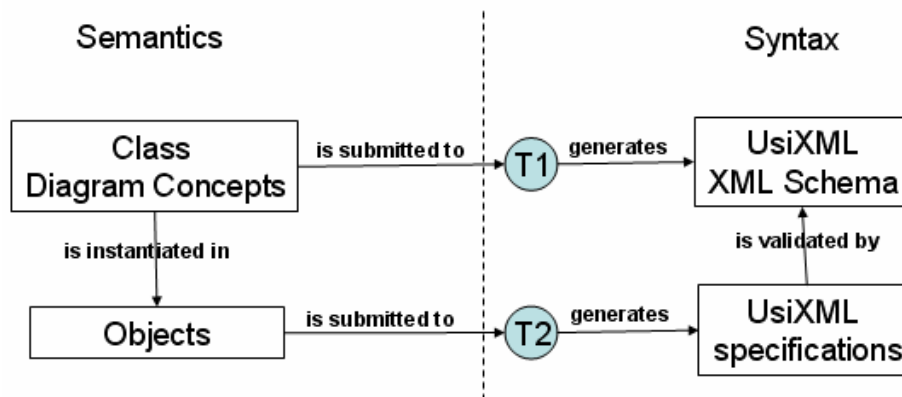


Figure 3-16 Generation of UsiXML specification

In the following figures we illustrate how instances of a set of class diagram concepts are submitted to transformations T2 in order to obtain UsiXML specification.

- A *class* becomes an *XML element* and *class attributes* become *XML attributes*: Figure 3-17 exemplifies how an instance of the *vocalMenu* class is mapped into an XML element with the associated attributes.

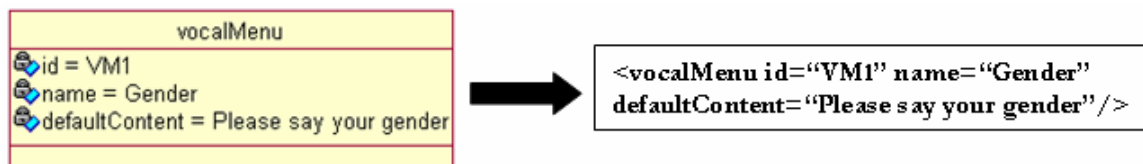


Figure 3-17 Transforming of a class to into UsiXML specification

- A *relationship class* and the associated *source/target classes* are transformed as follows: an *XML element* specifying the name of the relationship and *source* and *target XML elements* corresponding to the source and the target of relationship, respectively. Figure 3-18 exemplifies how a *graphicalTransition* relationship between two elements (i.e., a source represented by a *button* and a target represented by a *window*) is transformed into UsiXML specification.

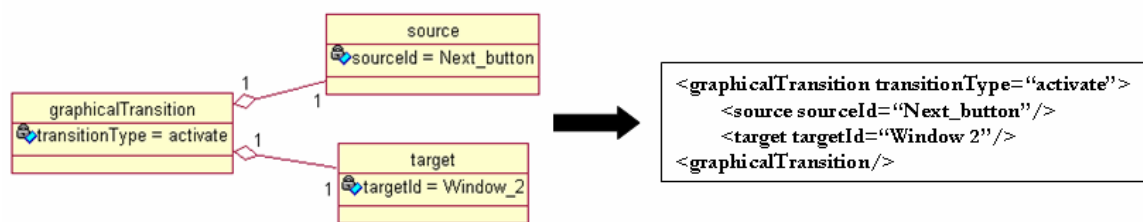


Figure 3-18 Transformation of a relationship class into UsiXML specification

- *Inheritance relationship class* is transformed into an *XML element* for which the value of the *type* attribute takes the name of the subclass. In addition, the attributes of the subclass become XML attributes of the created element. Figure 3-19 presents two objects of two different classes (i.e., *input* and *output*) that inherit attributes from the same superclass (i.e., *facet*). For each object an XML element named *facet* is created. The attributes of the subclass instances (i.e., the *inputDataType* and *outputContent*) become XML attributes of the corresponding *facet* element.

3. Conceptual Modeling of Multimodal User Interfaces

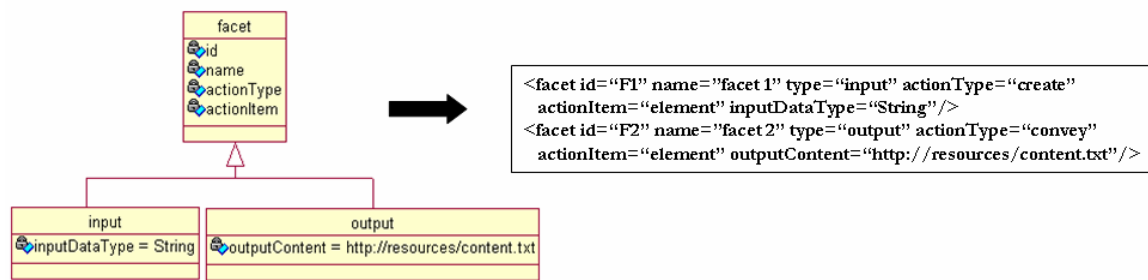


Figure 3-19 Transformation of the inheritance relationship into UsiXML specification

- Aggregate relationship* corresponds to an XML structure where the client class and the supplier class are transformed into XML elements according to the example provided in Figure 3-17. The XML element generated from the client class embeds the XML element generated from the supplier class. Figure 3-20 exemplifies how an instance of a client class (i.e., *vocalMenu*) and two instances of a supplier class (i.e., *vocalMenuItems*) are transformed into XML elements. The *vocalMenu* element will embed the two *vocalMenuItems* elements. UsiXML takes advantage of the XML document structure and allows to derive implicit relationships between objects. For instance, the structure of UsiXML syntax allows to infer two *vocalContainment* relationships: the *vocalMenu* VM1 embeds the *vocalMenuItems* VMI1 and VMI2, respectively.

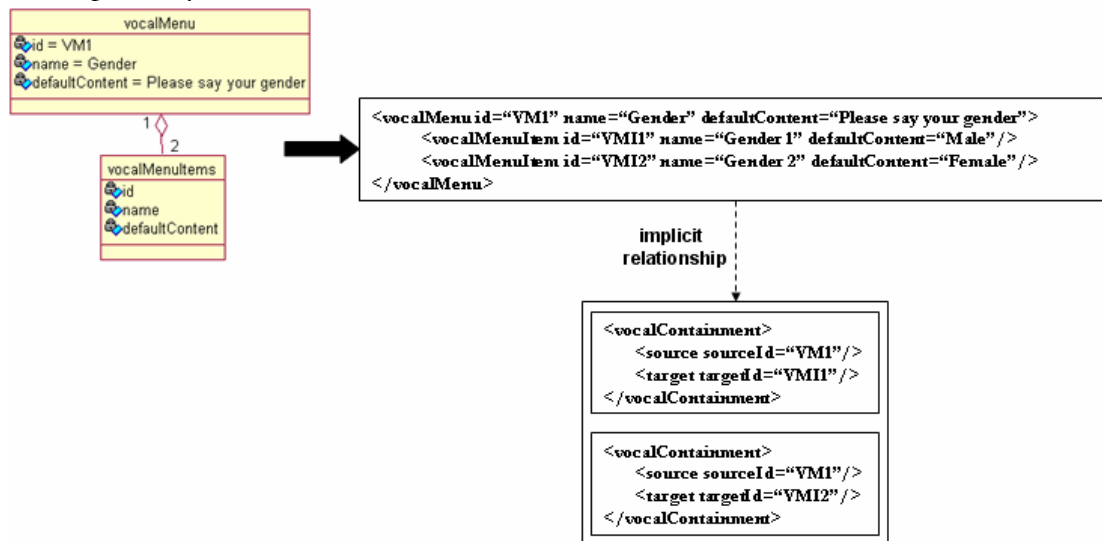


Figure 3-20 Transformation of the aggregation relationship into UsiXML specification

3.5.2 Concrete Syntax of Interaction Objects

In this section we provide the UsiXML syntax for a series of graphical, vocal and MM CIOs. Our methodology aims to cover the CARE properties (Requirement 3. Support for CARE properties concerning the input/output modalities). The *Complementarity* and the *Redundancy in input* properties require the system to perform data fusion for input modalities or data fission for output modalities which are both out of the scope of the current thesis (Section 1.4.3). Therefore, only the *Assignment*, *Equivalence* and *Redundancy in output* properties will be addressed.

3. Conceptual Modeling of Multimodal User Interfaces

Hereafter we present the possible input/output interactions for a label and a text field widgets that enable users to specify their names (second widget in Table 3-7):

- *Input:*
 - *Graphical interaction:* only graphical CIOs are involved.
 - *Vocal interaction:* only vocal CIOs are involved.
 - *Multimodal interaction with vocal assignment:* synchronization between VICs and GICs is required. In addition, *isEditable* attribute of the GIC set to *false* in order to disable the graphical input.
 - *Multimodal interaction with graphical and vocal equivalence:* synchronization between VICs and GICs is required. In addition, *isEditable* attribute of the GIC is set to *true* in order to allow the graphical interaction.
- *Output interactions:*
 - *Graphical interaction:* only graphical CIOs are involved.
 - *Vocal interaction:* only vocal CIOs are involved.
 - *Multimodal interaction with graphical and vocal redundancy:* both graphical and vocal CIOs are involved.

The input and output interactions identified above are combined together in Table 3-8 in order to identify a valid set of possible interactions for the considered task.

Input \ Output	G	V	MM with V assignment	MM with G and V equivalence
G	G	-	MM with V assignment in input and G assignment in output	MM with equivalence in input and G assignment in output
V	-	V for input and V for output	-	-
MM with G and V redundancy	MM with G assignment in input and redundancy in output	-	MM with V assignment in input and redundancy in output	MM with equivalence in input and redundancy in output

Table 3-8 Possible combinations of input/output interaction types for a label and a textFiled

For the label and a text field widget each valid combination in the above table is specified hereafter according to the UsiXML syntax. For the rest of the widgets in Table 3-7 the specifications is provided in Appendix C.

- *Graphical interaction:*

```
<box id="b1" name="Box 1"...>
  <outputText id="OT1" name="Output 1" defaultContent="Name".../>
  <inputText id="IT1" name="Input 1" isEditable="true" currentValue="$x".../>
</box>
```
- *MM with G assignment in input and redundancy in output:*

```
<box id="b1" name="Box 1"...>
  <outputText id="OT1" name="Output 1" defaultContent="Name".../>
  <imageComponent id="IC3" name="keyboard_icon" defaultContent="keyboard.jpg".../>
  <inputText id="IT1" name="Input 1" isEditable="true" currentValue="$x".../>
```


3. Conceptual Modeling of Multimodal User Interfaces

```
<imageComponent id="IC4" name="speaker_icon" defaultContent="speaker.jpg" .../>
</box>
```

```
<vocalForm id="VF1" name="Form 1" ...>
  <vocalFeedback id="F1" name="Feedback 1" defaultContent="Your name is $y" .../>
</vocalForm>
```

```
<synchronization>
  <source sourceId="F1"/>
  <target targetId="IT1"/>
</synchronization>
```

- *Vocal interaction:*

```
<vocalForm id="VF1" name="Form 1" ...>
  <vocalPrompt id="VP1" name="Prompt 1" defaultContent="Please say your name" .../>
  <vocalInput id="VI1" name="Input 1"/>
    <record id="RE1" name="Record 1" defaultContent="rec_msg" .../>
  </vocalInput>
  <audio id="AU1" name="Audio 1" defaultContent="Your name" audioSource="rec_msg"/>
</vocalForm>
```

- *MM with V assignement in input and G assignement in output:*

```
<box id="b1" name="Box 1" ...>
  <outputText id="OT1" name="Output 1" defaultContent="Name" .../>
  <imageComponent id="IC1" name="microphone_icon" defaultContent="microphone.jpg" .../>
  <inputText id="IT1" name="Input 1" isEditable="false" currentValue="x" .../>
</box>
```

```
<vocalForm id="VF1" name="Form 1" ...>
  <vocalPrompt id="VP1" name="Prompt 1" defaultContent="Please say your name" .../>
  <vocalInput id="VI1" name="Input 1" .../>
    <record id="RE1" name="Record 1" defaultContent="rec_msg" .../>
  </vocalInput>
</vocalForm>
```

```
<synchronization>
  <source sourceId="VI1"/>
  <target targetId="IT1"/>
</synchronization>
```

- *MM with V assignement in input and redundancy in output*

```
<box id="b1" name="Box 1" ...>
  <outputText id="OT1" name="Output 1" defaultContent="Name" .../>
  <imageComponent id="IC1" name="microphone_icon" defaultContent="microphone.jpg" .../>
  <inputText id="IT1" name="Input 1" isEditable="false" currentValue="x" .../>
  <imageComponent id="IC4" name="speaker_icon" defaultContent="speaker.jpg" .../>
</box>
```

```
<vocalForm id="VF1" name="Form 1" ...>
  <vocalPrompt id="VP1" name="Prompt 1" defaultContent="Please say your name" .../>
  <vocalInput id="VI1" name="Input 1" .../>
    <record id="RE1" name="Record 1" defaultContent="rec_msg" .../>
  </vocalInput>
  <audio id="AU1" name="Audio 1" defaultContent="Your name" audioSource="rec_msg"/>
</vocalForm>
```

```
<synchronization>
  <source sourceId="VI1"/>
  <target targetId="IT1"/>
</synchronization>
```

- *MM with equivalence in input and G assignement in output*

```
<box id="b1" name="Box 1" ...>
```

3. Conceptual Modeling of Multimodal User Interfaces

```
<outputText id="OT1" name="Output 1" defaultContent="Name" .../>
<imageComponent id="IC1" name="microphone_icon"
defaultContent="microphone.jpg" .../>
<imageComponent id="IC1" name="keyboard_icon" defaultContent="keyboard.jpg" .../>
<inputText id="IT1" name="Input 1" isEditable="true" currentValue="x" .../>
</box>

<vocalForm id="VF1" name="Form 1" ...>
  <vocalPrompt id="VP1" name="Prompt 1" defaultContent="Please say your name" .../>
  <vocalInput id="VI1" name="Input 1" .../>
    <record id="RE1" name="Record 1" defaultContent="rec_msg" .../>
  </vocalInput>
</vocalForm>

<synchronization>
  <source sourceId="VI1"/>
  <target targetId="IT1"/>
</synchronization>

▪ MM with equivalence in input and redundancy in output
<box id="b1" name="Box 1" ...>
  <outputText id="OT1" name="Output 1" defaultContent="Name" .../>
  <imageComponent id="IC1" name="microphone_icon"
defaultContent="microphone.jpg" .../>
  <imageComponent id="IC1" name="keyboard_icon" defaultContent="keyboard.jpg" .../>
  <inputText id="IT1" name="Input 1" isEditable="true" currentValue="x" .../>
</box>

<vocalForm id="VF1" name="Form 1" ...>
  <vocalPrompt id="VP1" name="Prompt 1" defaultContent="Please say your name" .../>
  <vocalInput id="VI1" name="Input 1" .../>
    <record id="RE1" name="Record 1" defaultContent="rec_msg" .../>
  </vocalInput>
  <audio id="AU1" name="Audio 1" defaultContent="Your name" audioSource="rec_msg"/>
</vocalForm>



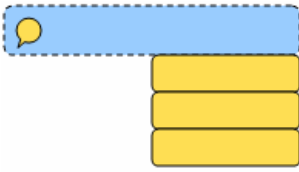




<synchronization>
  <source sourceId="VI1"/>
  <target targetId="IT1"/>
</synchronization>
```

3.6 Stylistics of interaction objects

The objective of this section is to provide a representation of several vocal objects composing our ontology in order to: (1) ease the exemplification of the different vocal concretizations of the design option values composing our design space (Section 4.2) and (2) facilitate their perception and manipulation when employed by future developed tools.

Typically, the stylistics of objects can take different forms (e.g., graphical, textual). In this dissertation we adopted a graphical representation for which we provide a justification of its components in Table 3-9.

3. Conceptual Modeling of Multimodal User Interfaces

Vocal CIO	Graphical representation	Representation rationale
<i>vocalGroup</i>		As it acts as a basic container the representation is composed of a dashed rectangle to suggest the containment purposes and a callout symbol to indicate the vocal character.
<i>vocalForm</i>		As it is a container that enables the dialog between the user and the system the representation is composed of a dashed rectangle to suggest the containment intentions, a user and a system icon next to a callout symbol.
<i>vocalMenu with vocalMenuItems</i>		As it is a container that enables to select among different options and by analogy with the menu provided by the graphical toolkits the representation is composed of a blue dashed oval to suggest the containment purposes, an overlaying callout symbol to indicate the vocal aspects and yellow ovals to indicate the vocal options.
<i>vocalPrompt</i>		As it is a system output that provides users with prompt information (usually concretized in questions) the representation is composed of a system icon and a callout symbol containing a question sign.
<i>vocalInput</i>		As it supposes that the system gathers input from the user by speech recognition or audio recording the representation is composed of a user icon next to a callout symbol and a system icon.
<i>vocalNavigation</i>		As it is used by the system to ensures the dialog transfer between vocal elements the representation is composed of a system icon next to a callout symbol positioned above a unidirectional arrow suggesting the transfer.
<i>audio</i>		As it is employed to play audio prerecorded files the representation is composed of a loudspeaker icon next to a musical note symbol.
<i>submit</i>		As it is used to send data to the server and/or to ensure the dialog transfer the representation is composed of a computer

3. Conceptual Modeling of Multimodal User Interfaces





		icon suggesting the server adjacent with a green arrow indicating the data upload, both positioned above a unidirectional arrow indicating a potential dialog transfer.
<i>if</i>		As it conditions the system execution of certain parts of a document the representation is composed of a dashed rectagle suggesting the containment of the parts to execute that embeds a computer icon with a set of connected arrows symbolizing the different decisions out of which only one will be executed (i.e., the red one)
<i>break</i>		As it interrupts the execution of the dialog between the system and the user the representation is composed of the icon employed for the <i>vocalInput</i> over which a cross was applied.
<i>exit</i>		As it terminates the execution of the vocal application the representation is composed of an open door icon and an arrow inviting to exit the room.

Table 3-9 Stylistics for several vocal concrete interaction objects

3.7 Conclusions

The current chapter presented an existing ontology that was extended in order to respond to the requirements of MM applications. In particular, a set of vocal and MM CIOs and the relationships between them have been introduced along with their semantics and stylistics. UsiXML, the UIDL selected to support our model based approach describes the syntax of the object composing the ontology. In Chapter 4 this ontolgy will serve as support for the design space based-method adopted in the current thesis.

4 A Transformational Method for Producing Multimodal User Interfaces

4.1 Introduction

After describing in Chapter 3 the ontology of our MM framework, Chapter 4 focuses on the transformational method applied over the previously introduced models. Section 4.2 introduces the design space along with the composing design options that aims to guide the designer's decisions during the development life cycle. Section 4.3 introduces the details of the selected model-to-model transformational approach and emphasizes its expansion thanks to the introduction of colored transformation rules. In Section 4.4 the 4 steps of the transformational approach are decomposed into sub-steps for which the corresponding design options supporting them are identified and exemplified.

4.2 Design space for user interfaces

4.2.1 Definition of design space

The capabilities of MM applications are well delineated since they are mainly constrained by what their underlying language offers, as opposed to hand-made MM applications. As the experience in developing such MM applications is growing, the need arises to identify and define major design options of such applications to pave the way to a structured development life cycle. Any software development life cycle should naturally evolve from early requirements to detailed ones, until a final system is developed and deployed. This evolution inevitably goes through identifying, defining, analyzing, comparing, and deciding between different, potentially contradictory, alternatives that may affect the entire process. The UI of this software does not escape from the aforementioned observations [Pala03].

We consider that a *design option* represents a design feature which effectively and efficiently supports the progress of the development life cycle towards a final system while ensuring some form of quality. For each design option, a finite set of design option values denotes the various alternatives to be considered simultaneously when deciding in favour of a design option. For instance, a designer confronted with a design decision concerning the presentation of the UI typically selects among a set of alternatives such as: present all the information in one window (e.g., if the UI is to be rendered on a desktop PC), separate the information in several different windows presented sequentially (e.g., if the UI is to be rendered on a PDA with reduced screen size capabilities) or render the information vocally to users employing mobile phones.

4. A Transformational Method for Producing Multimodal User Interfaces

Design options often involve various stakeholders representing different human populations (e.g., the end users, the marketing and the development team) with their own preferences and interests in the development life cycle. When a particular design option value is assigned to a design option, it is considered that a *design decision* is taken. These decisions often result from a process where the various design options are gathered, examined and ranked until an agreement is reached among stakeholders. The decision process is intrinsically led by consensus since stakeholders' interests may diverge and by trade-off between multiple criteria, which are themselves potentially contradictory.

Therefore, we define a design space as:

A structured combination of design options having assigned a finite set of design option values that support the stakeholder's design decisions during the development life cycle of multimodal user interfaces.

The design space analysis [Limb00] represents a significant effort to streamline and turn the open, ill-defined and iterative [Rous05] interface design process into a more formalized process structured around the notion of design option. A design space consists of an n-dimensional space where each dimension is denoted by a single design option. For this space to be orthogonal, all dimensions, and therefore all their associated design options, should be independent of each other. This does not mean that a dimension cannot be further decomposed into sub-dimensions, case in which the design space becomes a snowflake model.

4.2.2 Rationale for choosing a design space

4.2.2.a Advantages

Design options present several important advantages:

- When they are explicitly defined, they clarify the development process in a structured way in terms of options, thus requiring less design effort and striving for consistent results if similar values are assigned to design options in similar circumstances.
- Defining a design option facilitates its incorporation in the development life cycle as an abstraction which is covered by a software, perhaps relying on a model-based approach. Ultimately, every piece of development should be reflected in a concept or notion which represents some abstraction with respect to the code level as in a design option. Conversely, each design option should be defined clearly enough to drive the implementation without requiring any further interpretation effort. For example, the design option concerning the presentation of the UI will result in the generation of a set of widgets that satisfy the corresponding constraints.
- The adoption of a design space supports the tractability of more complex design problems or for a class of related problems.

4.2.2.b Shortcomings

The design space suffers from set of shortcomings as the design options could be very numerous, even infinite in theory. But in practice, it is impossible to consider a very large amount of design options because of several reasons:

- They are too complex or expensive to implement.
- They do not necessarily address users' needs and requirements.
- They are outside the designer's scope of understanding, imagination or background.
- Their decision is not always clear and when it is taken it may violate some usability principles or guidelines. For example, deciding a particular design option may lead to a design which is probably feasible to be implemented, but which is likely to be unusable or inconsistent. Reducing a design to a set of design options may restrict the designers' creativity or could be perceived as such. Design options, anyway and anyhow, will always represents a restriction of the complete design space, the problem being to identify the relevant ones and leaving out the ones that are too detailed and that do not affect the UI quality.
- Not all design options could be discovered or defined in an independent way as they sometimes appear very intertwined. Moreover, not all the possible values of a design option may be equal in implementation cost.

4.2.2.c Justification

We consider important to define such a design space for the development of MM applications (*Requirement 8: Approach based on design space*) because of the qualities it ensures:

- Intrinsic qualities: according to [Beau00] a design space is by its nature:
 - *Descriptive*: all design options are documented and allow summarizing any design in terms of design options values. These values have been identified and defined based on observation and abstraction of UIs and by introspection over the personal knowledge regarding the information systems.
 - *Comparative*: several different designs of MM UIs may be analyzed and compared based on the design options considered in their development so as to assess the design quality in terms of factors like utility, usability, portability, etc.
 - *Generative*: the design space allows to discover potentially new values for the existing design options or to introduce new design options associated with yet under explored design aspects.
- Extrinsic qualities:
 - Independently of any implementation or tool support, having at hands a design space where a small, but significant set of design options could be envisaged is a contribution which could be useful to any designer of MM applications. This provision helps designers to avoid to replicate the identification and definition of these design options, while leaving them free to consider other options or to overwrite the existing ones.
 - The languages in which they are implemented restrict the amount of possible interfaces to obtain as they directly set the CARE properties to assignment, equivalence, complemetarity or redundancy. Moreover, the interaction styles [Beau00] supported by these languages make them appropriate for certain types

4. A Transformational Method for Producing Multimodal User Interfaces

of applications (e.g., information systems), but totally inadequate for other types (e.g., air traffic control) [Mac189]. Therefore, a design space composed of design options independent of the interaction modality is a solution that offers designers explicit guidance during the development life cycle by providing flexibility with respect to the interaction modalities to select and the types of combination to set among them.

- Multimodal applications often employ graphical and vocal interaction modalities. When used together they multiply the combinations of modality concretizations assigned to design options values thus complexifying the entire design space. Sometimes, a design option which was estimated relevant for a particular modality (e.g., the graphical modality) may become totally irrelevant for other modalities (e.g., the vocal modality) or for its combination with other modalities (e.g., multimodality with graphical and vocal interaction). Therefore, a design space defined with an explicit set of design options and values clarifies the development process and simplifies the design decisions.

4.2.3 Design options for user interfaces

Our design space (Figure 4-1) is composed of a set of sixteen modality independent design options introduced hereafter according to a structured schema: each design option is consistently named, defined, justified through a rationale, and explicated with design option values. The design options will be specified in rectangles, whereas their associated values in ovals. Each value is sustained by examples that illustrate its concretization for the following interaction modalities:

- *Graphical*: supported by UsiXML graphical CIOs/group of objects.
- *Vocal*: supported by UsiXML vocal CIOs/group of objects for which the stylistics introduced in Section 3.6 is used.
- *Multimodal*: combining together the previously specified interactions taking into consideration the CARE properties.

Two particular situations have been identified: (1) a design option value does not necessarily have a correspondent concrete object for all considered interactions; (2) the same concrete object can be associated to multiple values.

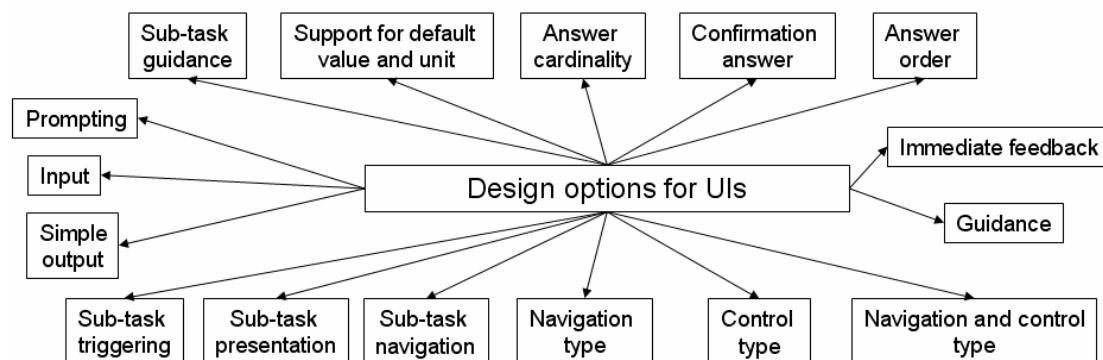


Figure 4-1 The design options composing the design space

(1) Sub-task triggering

Definition. It specifies which entity has the initiative to launch and to control the triggering of the sub-tasks.

4. A Transformational Method for Producing Multimodal User Interfaces

Rationale. When considering the interaction between the user and the system as a dialog between partners, it is important to consider which partner has the initiative in the conversation. The system can initiate all dialogs, in which case the user simply responds to requests for information or action. This dialog is called system-driven because the system more or less decides which action the user may perform [Gram96]. Alternatively, the system might react only to user input, in which case the dialog is called user-driven because the user has more freedom in choosing the next action. Consequently, the initiative could be also mixed, if either the user or the system may have the initiative.

System-driven dialogs tend to be preemptive as they limit the user's choice of next available action, whereas user-driven interaction favors non-preemptiveness. For instance, a dialog box may prevent the user from interacting with the system in any way that does not have a direct input to the box, thus preventing user's flexibility. In general, designers want to minimize the system's ability to preempt the user although some situations may require it for safety reasons. For example, in critical systems in which a user error would result in serious damage without a chance for recovery, it is desirable, or even necessary, to limit the user's freedom. [Lars06] shows that in MM UIs a user-driven dialog and a mixed one may be useful for experienced users, while they are confusing and inhibiting for novice users.

Consequently, the designer must have a good understanding of the set of tasks to perform, how those tasks are interrelated and the experience level of the users with those tasks in order to minimize the likelihood that the users will be prevented from initiating or advancing some tasks at the time they want to do so.

Values. Even if we identified three types of dialog initiatives, the mixed one is not considered a possible value as it can be obtained as a result of the combination of the user and the system initiative:

- *System:*
 - a GUI that automatically changes the focus on the next widget once the user finished to fill in the current one.
 - in VUI:
 - System: *"What is your zip code?"*
 - User: *"1348"*.
 - System: *"Please say your gender."*
 - User: *"Female"*.
- *User:*
 - a GUI that enables the user to choose the widget to fill in at any time.
 - in VUI:
 - System: *"What information would you like to input?"*
 - User: *"Gender"*.
 - System: *"Please say your gender."*
 - User: *"Female"*.

(2) Sub-task presentation

Definition. It specifies the way in which the system is presenting the sub-tasks to the user.

Rationale. Designers of MM UIs attempt to take into account the relation between the structure of the tasks to be performed and the manner of presenting it in the UI. Central

4. A Transformational Method for Producing Multimodal User Interfaces

to this design issue is the way in which the information is conveyed and cognitively processed by the end users [Norm86].

If we consider the GUIs, the most obvious and apparent mode of presenting the information is the physical layout of the display and its functional properties. Here the user visualises UI screens containing for instance windows, dialog boxes, selection lists, tabbed dialog boxes, etc. and information displayed thereon. Design issues at this level include, but are not restricted to, the amount of information that can be presented on the screen, the spatial location of the screens, the linking of information from one screen to another.

Therefore, in order to expand the amount of information presented to the user, the physical layout is being split into small chunks according to the structure of the tasks to be performed, thus creating multiple screens. These screens could be separate (e.g., separated windows) or gathered together on the same screen (Figure 4-2). Furthermore, the later could be overlaid or fused one next to the other. With overlaid screens, there is no increase in the real amount of information displayed at once. However, there is an apparent increase in the sense that the users infer that the information is there, although it is covered up. A real increase in the amount of information is achieved only with fused screens.

In order to expand the utility of the physical layout a number of methods for coordinating the screens have been proposed and developed [Shne86]. Therefore, a UI may be structured so that if the user is selecting an item on a working screen, instances of this item and/or detailed information about the item could be displayed on another screen without changing the contents on the working screen.

After interpreting and interacting with the physical layout, the user infers from the system a cognitive model of what is going on. The result will be a cognitive structure in the user's mind with elements and relationships among them that map the elements and relationships at the UI level. Therefore, the designer should model the layout not only for efficient communication of information, but also to induce and suggest an appropriate cognitive layout. One cognitive representation that has a very powerful visual impact is the zoom in - zoom out screen. A set of selection items are displayed one below the other and once an item is selected it will zoom in the corresponding information by unrolling a screen that finds its place between the selected item and the one right below it. A zoom out concretized in rolling up the screen is possible at any time either by selecting the current item or any other item in the selection list.

The human perceptual system tends to group objects that are in close proximity or similar in size, shape, color or orientation. Therefore, different screens fused together should visually organize information displayed thereon. The organization should take into account the topology (location) and some graphical characteristics (format) in order to indicate the relationships between the various information displayed and whether they belong to a given class or not [Bast93]. The users' understanding of a screen depends, among other things, on the ordering, the positioning and the distinction of objects (e.g., images, text, commands) that are presented. Users will detect the different items or groups of items, and learn their relationships more easily if, on the one hand, they are presented in an organized manner (e.g., alphabetic order, numbered order), and on the other hand, if the items are presented in formats that indicate their similarities or

4. A Transformational Method for Producing Multimodal User Interfaces

differences (e.g., a separation line between information belonging to different classes). Consequently, the learning and remembering of items will be improved. Moreover, grouping or distinction of items will lead to a better guidance for the end users.

VUIs are transient [Java 98] so the user doesn't rely anymore on the persistent visual support of the screens. Therefore, the temporal aspect will play an important role that has a powerful effect on the perceptual grouping and inferred order of information. To keep from overloading the user's short-term memory, information presented in a VUI must generally be more concise than information presented visually. Often, only the most essential information should be presented initially (e.g., the list of vocal items), with the opportunity for the user to access detailed information about these items at a lower level. By analogy with the traditional notion of graphical formatting presented above, we consider the vocal formatting as a useful technique that increases the bandwidth of vocal communication by using speech and non-speech cues to overlay structural and contextual information of vocal output. For this purpose [IBM03a] specifies a set of recommendations. For instance, bulleted list presented in GUI should have as equivalent in VUI a short sound snippet as an auditory icon at the beginning of each item in the list, graphical bold or italics used to introduce an ordered list of sub-task titles should be emphasised in VUIs using auditory inflection technique such as changing volume or pitch. Moreover, for grouping or separating a set of vocal items whether they belong to the same class or not, designers should consider audio files playing a unique sound or special tone.

Values. The possible values identified as leaves in Figure 4-2 are by no means exhaustive and represent a point of departure in the system design meant to stimulate further thinking about how the user interprets the information presented by the system.

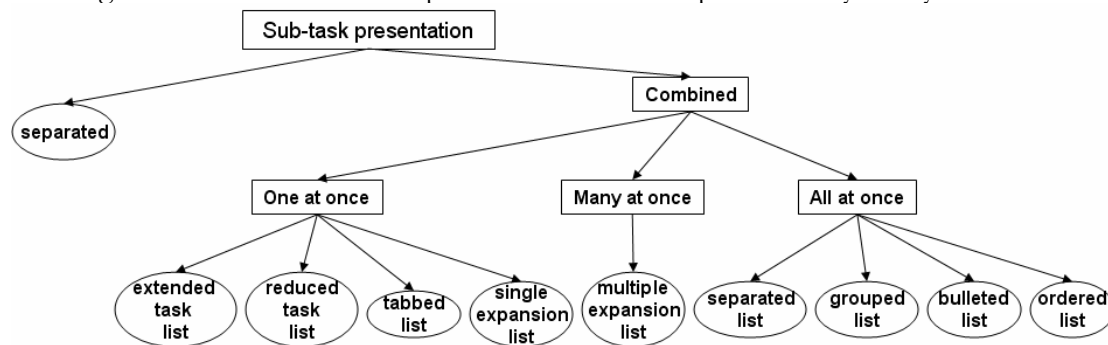


Figure 4-2 Sub-task presentation values

For each value a correspondent vocal and/or graphical concrete object is illustrated Figure 4-3. Thus, the presentation of the sub-tasks can be:

- *Separated*: each sub-task is conveyed in different containers (e.g., different screens each one containing a *window* in GUIs or different *vocalGroups* in VUIs).
- *Combined*: the sub-tasks are conveyed in the same container (e.g., a single screen containing a *window* or a *dialog box* in GUIs or a *vocalGroup* in VUIs):
 - *One at once*: only one sub-task is presented at a time. The possible values are:
 - ✓ *Extended task list*: in GUIs a selection screen (on the left) containing the available *items* and their associated overlaid screens presenting the detailed

4. A Transformational Method for Producing Multimodal User Interfaces

information (on the right), whereas in VUIs a *vocalMenu* with *vocalMenuItems* indicating the name of the possible *vocalGroups* among which to choose.

- ✓ *Reduced task list*: in GUIs a *comboBox* containing the available *items* to select and their associated overlaid screens presenting the detailed information, whereas in VUIs a *vocalForm* composed of a *vocalPrompt* inviting the user to select an option and a *vocalInput* regarding the user's options that are further specified in a *vocalMenu* with *vocalMenuItems*.
- ✓ *Tabbed list*: in GUIs a *tabbedDialogBox* with *tabbedItems* in its upper part and associated overlaid screen presenting the detailed information, whereas in VUIs the concretization is the same as for extended task list counterpart.
- ✓ *Single expansion list*: in GUIs a *floatWindow* containing the *floatItems* that enables to zoom in the detailed information displayed in associated fused screens, whereas in VUIs the concretization is the same as for extended task list counterpart.
- *Many at once*: multiple sub-tasks are presented in the same time. The possible value is *multiple expansion list*. The graphical concretization is identical to the *single expansion list* counterpart, with the difference that multiple surfaces can be presented simultaneously. We consider the vocal concretization difficult to achieve due to temporal constraints.
- *All at once*: all sub-tasks are presented simultaneously. The possible values are:
 - ✓ *Separated list*: in GUIs the separation can be ensured by a blank space or an horizontal line, whereas in VUI the *vocalPrompts* synthesizing the sub-tasks are separated by *audio* elements playing audio files.
 - ✓ *Grouped list*: in GUIs the separation can be ensured by a frame line or colored background, whereas in VUIs the concretization is the same as for separated list counterpart.
 - ✓ *Bulleted list*: in GUIs the tasks are introduced by a ●, □, →, √, etc., whereas in VUIs *audio* elements playing sounds such as beeps, dongs, etc. are followed by *vocalPrompts* synthesizing the sub-tasks name using auditory inflection techniques.
 - ✓ *Ordered list*: in GUIs the sub-tasks are ordered by items such as (1, 2, 3), (a, b, c), (i, ii, iii), (α, β, γ), etc., whereas in VUIs *vocalPrompts* synthesizing the items specified in the graphical example are followed by *vocalPrompts* uttering the sub-tasks name using auditory inflection techniques.

4. A Transformational Method for Producing Multimodal User Interfaces

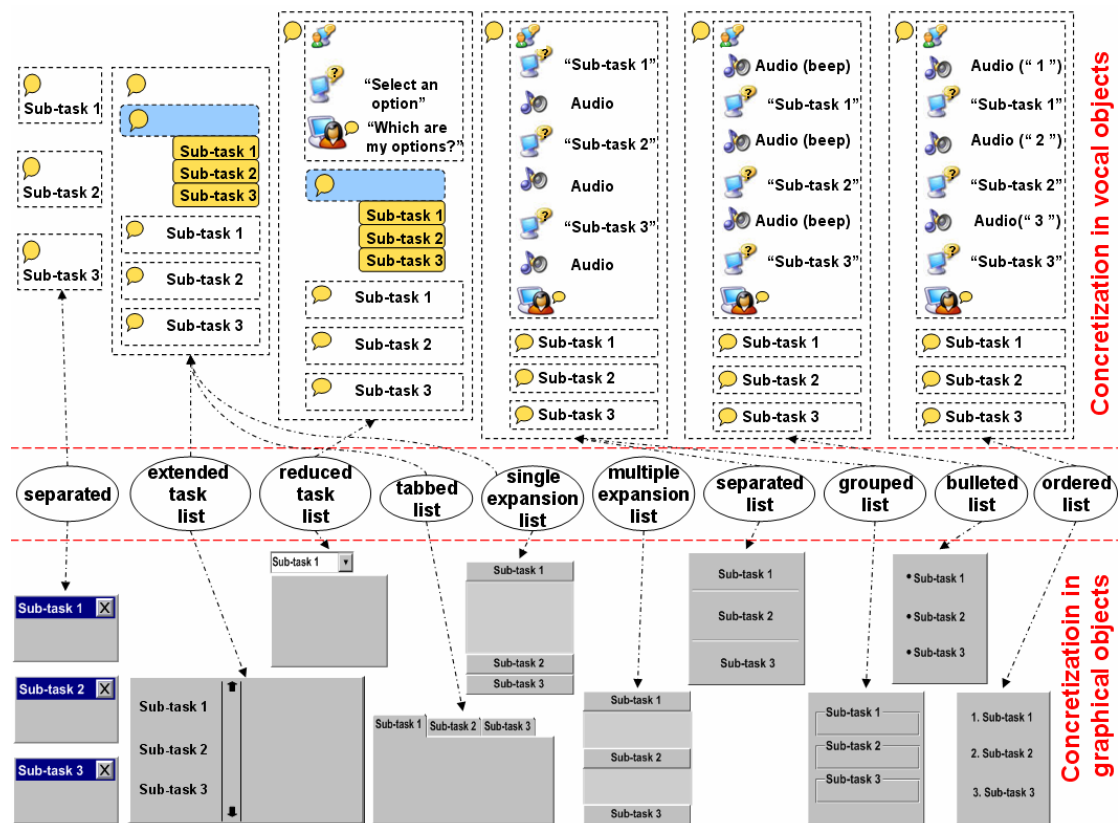


Figure 4-3 Sub-task presentation values and a possible concretization in vocal and graphical objects

Whenever the designer needs to convey sibling sub-tasks into different presentations, an approach by problem reduction is applied, i.e. the simplification to an already solved problem. This is done by transforming the initial task structure into an equivalent one where the sibling sub-tasks are placed one level lower and are grouped as sub-tasks of a newly introduced task.

Sub-task presentation is a design option that is independent of the type of temporal relationships specified between the sub-tasks. Even if all the current presentations are valid there are some better suited than others according to IFIP quality properties [Gram96]. For instance, the *extended task list* value is better than the *reduced task list* one which is at its turn better than the *separated* value. This is due to the fact that the former value makes the sub-tasks observable and browsable, the second it is just browsable, whereas the later is neither observable nor browsable.

(3) Sub-task navigation

Definition. It specifies the manner in which the user is able to browse the sub-tasks presented in a UI.

Rationale. The navigation scheme of a UI should allow users to find and access information effectively and efficiently when interacting with the system. Navigation is characterized in [Gram96] by the reachability property which refers to the possibility of navigating through different system states. This property contributes to the system quality and must be considered explicitly during the development life cycle. It can be

4. A Transformational Method for Producing Multimodal User Interfaces

defined at any level of detail, but our interest is focused mainly on the observable states. One of the issues of the reachability property is whether the user can navigate from any given observable state to any other observable state. From the user's point of view it may be useful to distinguish between backward and forward reachability. The user may want backward reachability in order to get back to some previous state of the interaction, after having made a mistake or realizing a need for some previous information. Forward reachability means that the user is able to proceed to any desired interaction state, independently of previous dialog development.

Synchronization is an issue specific to the MM environment. Since the MM applications are using navigational scheme that involve both vocal and graphical modalities, they should always be synchronized [IBM03b]. For example, while the vocal side of the application is processing a dialog assigned to sub-task one, the graphical counterpart should not be directed to the second sub-task unless the vocal side is also making a transition to that sub-task. If the user forces the transition manually, the VUI should adjust accordingly by either stopping the process for sub-task one and starting the process for sub-task two or by playing an error message. The graphical UI should display the page to match the vocal response as well.

In Section 3.4.4 we extended the *graphicalTransition* and *vocalTransition* relationships with the *activation* and *deactivation* values. Hereafter we propose a graphical notation assigned to these values that comes as an extension of [Vand03] dedicated to the navigation between interaction objects. Thus, the deactivation actions are symbolized with a yellow bulb, whereas the activation actions are illustrated with a red bulb.

Values. Two design option values were identified, for each one the correspondent vocal and graphical concrete objects being illustrated (Figure 4-4):

- *Sequential (synchronous)*: enables users to navigate forward and backward in a linear manner from the current sub-task to a neighbour sub-task only (e.g., in the GUI is ensured by the group of (*PREV*, *NEXT*) *buttons*, whereas in the VUI is ensured by *vocalNavigation* objects).
- *Asynchronous*: enables users to navigate forward and backward in a linear or non-linear manner from the current sub-task to any other desired sub-task, thus providing users with more flexibility in manipulating the interface (e.g., in the GUI is ensured by *buttons* specifying the possible sub-tasks to visit, whereas in the VUI is ensured by *vocalNavigation* objects).

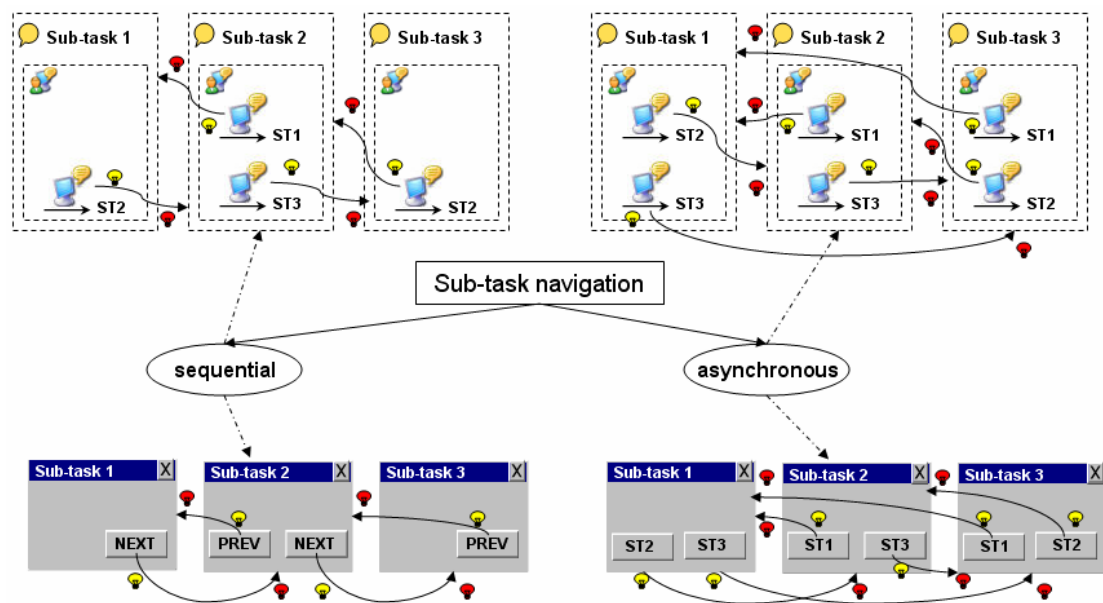


Figure 4-4 Types of navigation between sub-tasks and a possible concretization in vocal and graphical objects

(4) Navigation type

Definition. It specifies the type of *containment* and the *cardinality* of the objects/logically grouped set of objects that ensure the navigation (Figure 4-5).

Rationale. The containment refers to the position of the navigation objects with respect to the information presented by each sub-task. The navigation objects/group of objects could have an instance in each sub-task presentation or could be concretized in a general navigational object/group of objects that ensure the browsability among all sub-tasks. The cardinality specifies the number of objects that ensure the navigation. The traditional design of a user interface considers a single object/group of objects that ensure the navigation between the sub-tasks. However, two or more navigational objects/group of objects ensuring the same functionality could be conveyed by the system and made available to the user simultaneously. These objects might improve the interaction flexibility and reduce the access time to the navigational functionalities. However, their redundant character could mislead the user if they are not carefully selected and conveyed.

Values. From the *containment* point of view two values are identified:

- *Local:* each sub-task has a corresponding navigation object (e.g., in GUI a *button* embedded in its correspondent *groupBox*, whereas in VUI a *vocalNavigation* embedded in its correspondent *vocalGroup*).
- *Global:* all sub-tasks have one common navigation object (e.g., in GUI a *button*, whereas in VUI a *vocalMenu* with *vocalMenuItems* indicating the name of the possible *vocalGroups* to visit).

From the *cardinality* point of view two values are identified:

- *Simple:* a single object or a single logically grouped set of objects ensures the navigation (e.g., in the GUI a group of *tab items* of a *tabbedDialogBox*, whereas in the VUI a *vocalNavigation* object embedded in its correspondent *vocalGroup*).

4. A Transformational Method for Producing Multimodal User Interfaces

- Multiple:** two or more objects/logically grouped set of objects ensure the navigation (e.g., in the GUI a group of *tabItems* in a *tabbedDialogBox* and the (NEXT, PREVIOUS) group of buttons, whereas in the VUI a *vocalMenu* with *vocalMenuItems* indicating the name of the possible *vocalGroups* to visit at any time during the dialog and a *vocalNavigation* object embedded in its correspondent *vocalGroup*).

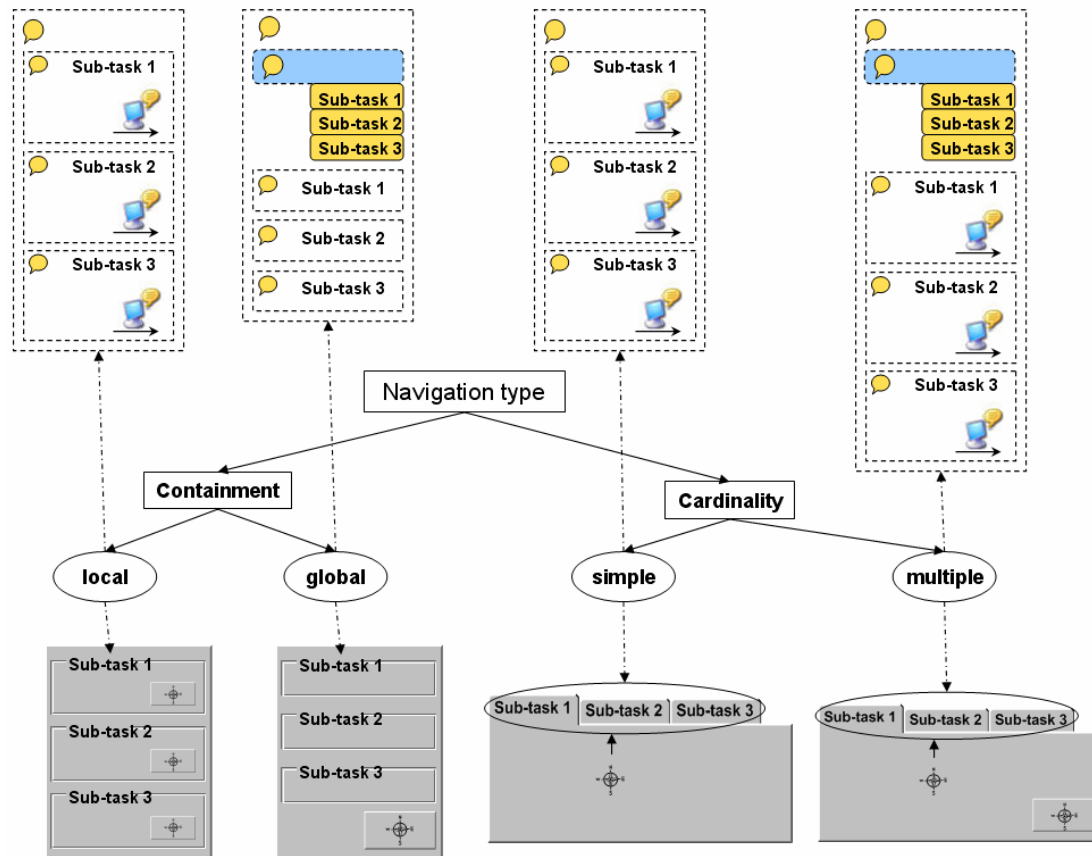


Figure 4-5 Navigation type values and a possible concretization in vocal and graphical objects

(5) Control type

Definition. It specifies the type of *containment* and the *cardinality* of the objects/logically grouped set of objects that ensure the data control (Figure 4-6).

Rationale. By data control we understand any operation in charge with the exchange of data between the user and the system (e.g., storing data into a database, retrieving data from a database, sending data to the system in order to be processed, canceling one of the previously specified operations). The containment refers to the position of the control objects with respect to the information presented by each sub-task. The data control objects/group of objects could have an instance in each sub-task presentation or could be concretized in a general object/group of objects that ensure the data control of the UI. The cardinality specifies the number of objects that ensure the data control. The traditional design of a UI considers a single object/group of objects that ensure the data control of the application. However, two or more control objects/groups of objects ensuring the same functionality could be conveyed by the system and made available to the user simultaneously. These objects might improve the flexibility of data control and

4. A Transformational Method for Producing Multimodal User Interfaces

reduce the access time to the data control functionalities. However their redundant character could mislead the user if they are not carefully selected and conveyed.

Values. From the *containment* point of view two values are identified:

- *Local*: each sub-task has a corresponding control object (e.g., in GUI a *button* embedded in its correspondent *groupBox*, whereas in VUI a *submit* object embedded in its correspondent *vocalForm*).
- *Global*: all sub-tasks have one common navigation object (e.g., in GUI a *button*, whereas in VUI a *submit* object).

From the *cardinality* point of view, two values are identified:

- *Simple*: a single object or a single logically grouped set of objects ensures the data control (e.g., in the GUI a group of *tabItems* of a *tabbedDialogBox*, whereas in the VUI a *submit* object embedded in its correspondent *vocalForm*).
- *Multiple*: two or more objects/logically grouped sets of objects ensure the data control (e.g., in the GUI a group of *tabItems* in a *tabbedDialogBox* and an *OK button*, whereas in the VUI a general *submit* object and for each sub-task a *vocalForm* embedding a *vocalPrompt* requesting the user to decide whether the data will be send to the server or not, a *vocalInput* gathering user's responses and an *if* object used to send the data thanks to the *submit* object if the condition holds true).

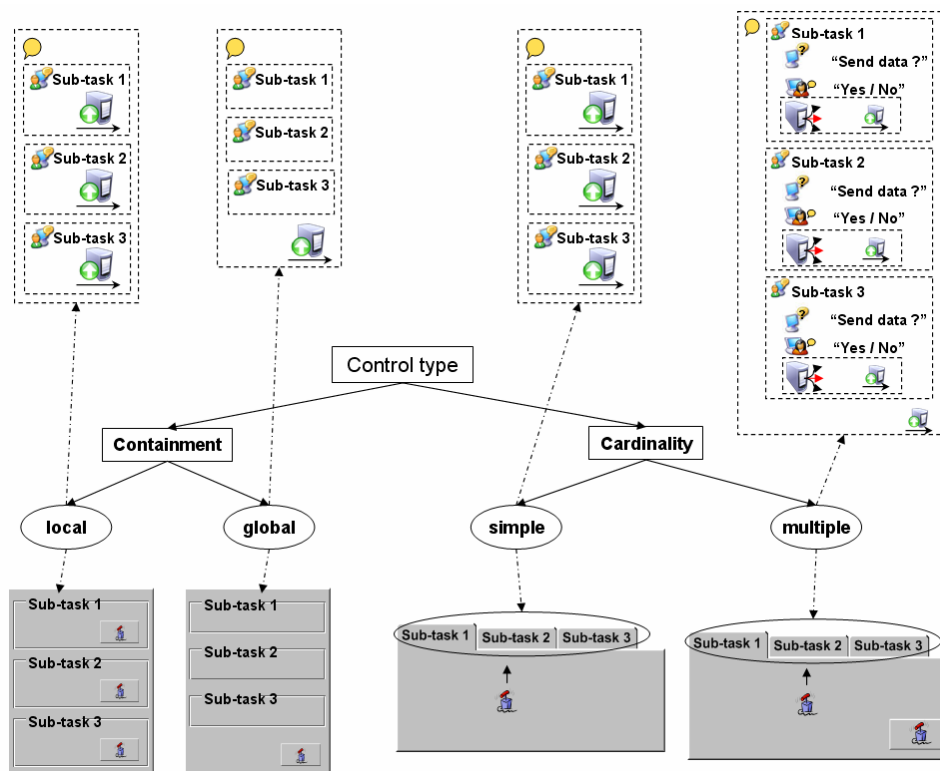


Figure 4-6 Control type values and a possible concretization in vocal and graphical objects

(6) Navigation and control type

Definition. It specifies whether the navigation and data control are ensured by the same object/logically grouped set of objects (Figure 4-7).

4. A Transformational Method for Producing Multimodal User Interfaces

Rationale. The navigation and the data control functionalities could be grouped together in order to be ensured by one single object thus improving the interaction speed between the system and the user. This design decision has its drawbacks in the sense that the designer should identify possible errors and recover the system according to the functionality that generated them. On the other side, a clear separation of the two functionalities could be ensured by the use of two different objects/groups of objects which differentiate semantically the triggered actions. The advantage is that the user is not confused anymore by the meaning of his/her actions.

Values. Two design option values were identified:

- *Separated:* different objects/logically grouped sets of objects ensure the control and the navigation between sub-tasks (e.g., in the GUI two buttons, one ensuring the navigation and one the data control, whereas in the VUI two vocal components, a *vocalNavigation* object ensuring the navigation and a *submit* object ensuring just the data control).
- *Combined:* the same object/logically grouped set of objects ensures simultaneously the navigation and the control (e.g., in the GUI the same button ensures the navigation and the control, whereas in the VUI the *submit* element ensures both the navigation (thanks to the *nextContainer* attribute) and the control).

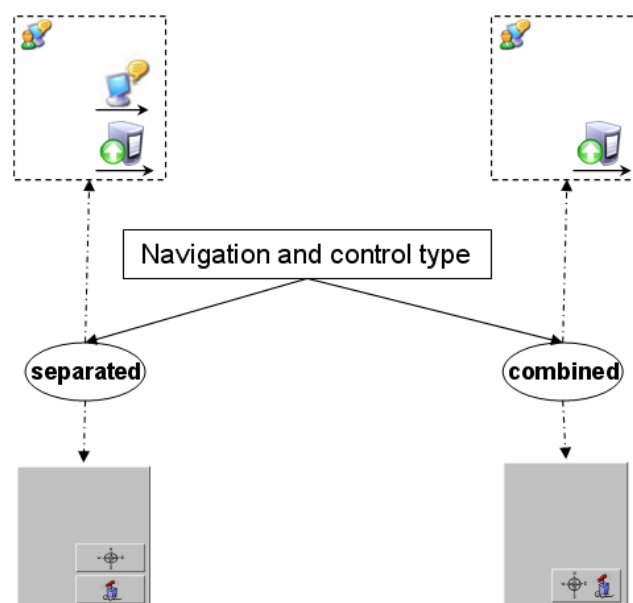


Figure 4-7 Navigation and control type values and a possible concretization in vocal and graphical objects

(7) Sub-task guidance

Definition. It specifies whether the end-user is guided with the possible answers to utter.

Rationale. As an application becomes more complex, offering the user more choices, the need of guidance becomes mandatory. For simple applications with fewer choices, the user may need guidance only the first time the application is run. Moreover, a novice user may not know the meaning of a field or the list of valid commands. For all these situations, the designer should convey the list of choices so that users can be guided with

4. A Transformational Method for Producing Multimodal User Interfaces

the possible options to select [Lars06]. However, if the user is likely to know the set of valid responses as it is obvious (e.g., the gender of a person) or if the list contains long or nearly infinite set of items, the designer shouldn't guide the user with the possible options to select.

Values. Two design option values were identified:

- *Guided*: the system provides the possible answers to the end-user
 - GUI that provides the user with the possible car colors to select (Figure 4-8).
 - VUI:
System: "Choose between green, red and black".
User: "Red".
- *Unguided*: the system doesn't provide the user with the possible answers:
 - GUI: in order to specify his/her name the user is not guided as there are numerous values that can be specified (Figure 4-9).
 - VUI:
System: "What is your name?"
User: "Michael".

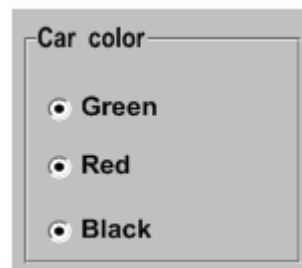


Figure 4-8 Guided sub-task in GUI



Figure 4-9 Unguided sub-task in GUI

(8) Support for default value and unit

Definition. It specifies whether the system provides default values to the user and the corresponding unit for data entry.

Rationale. The workload with respect to the number of actions necessary to accomplish a task should be one of the concerns of UI designers. This requires the limitation, as much as possible, of the number of steps users must go through when interacting with the UI. The more numerous and complex the actions necessary to achieve a task, the more the workload will increase and, consequently, the more probable the risk of making errors. For this purpose default values and their assigned units should be conveyed to the users that are dealing with data entry [Bast93]. This will contribute to the minimization of the number of actions to perform when accomplishing the task. However, default values and units do not always make sense. For data entries that involve a high number of choices providing a default value is not appropriated. Moreover, the default unit might not be necessary when it is obvious as it is frequently used on a day to day basis. Therefore, in these latter examples no default values or units should be provided as they will only influence in a negative manner the concision of the UI [Bast93].

Values. The possible values are:

- *With support*: the system provides default values and units

4. A Transformational Method for Producing Multimodal User Interfaces

- GUI: the user has to select the zoom level which is expressed on a scale from 10 to 500 and the unit is expressed in percentage (Figure 4-10).
- VUI: the vocal equivalent of the situation described above is:
System: "Please specify the zoom level on a scale from 10 to 500 percentage".
User: "75".
- *Without support:* the system provides neither the default values, as there is an infinite number of choices, nor the unit, as the data entry is not assigned to any metric
 - GUI: when the user has to provide his/her first name he/she will type it in the data entry field (Figure 4-11).
 - VUI: the vocal equivalent of the situation described above is:
System: "Please specify your first name".
User: "Michael".

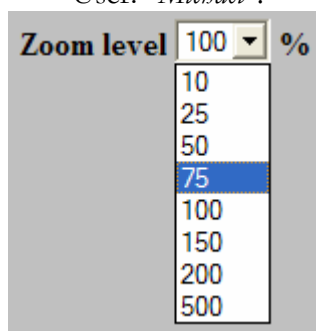


Figure 4-10 Ambiguous answer in GUI Figure 4-11 Unambiguous answer in GUI

(9) Answer cardinality

Definition. It specifies the cardinality of the items composing the user's answer.

Rationale. When interacting with the system, the user's input consist either of a single answer from an undefined range of values or a single/multiple selection of item(s) from a list of predefined options. In order to support these situations the research literature recommends specific interaction objects depending on the considered interaction modality. Thus, in GUIs a single input from a list of undefined options should be realized in an entry field [John95], whereas multiple inputs should be specified in an entry field and accumulated in a listbox [Boda94]. The same studies show that for predefined values in a list of less than seven values, radio buttons should be used for simple selection and check boxes for multiple selection, whereas for more than eight values, single selection listbox objects are recommended for simple selection and multiple selection listbox objects with accumulator for multiple selection. For VUIs the number of help requests and errors increases [Enge89] if more than four options are provided [Goul87, Knol90] or even more than three according to [Deva91]. Moreover, when the users are unfamiliar with the system, the number of options should be provided in order to cue them not to respond too soon if uncertainty exists [Schu92].

Values. The possible values are:

- *Simple:* a single answer from an undefined range of values or the selection of a single item from a list of predefined options:
 - GUI: when asked to fill in the year of birth there is only one value that can be selected (Figure 4-12).

4. A Transformational Method for Producing Multimodal User Interfaces

- VUI: the vocal equivalent of the situation described above is:
System: “Which is your year of birth?”
User: “1980”.
- *Multiple*: the selection of multiple items from a list of predefined options:
 - GUI: when asked to specify the his/her hobbies, the user may select multiple items (Figure 4-13).
 - VUI: The vocal equivalent of the situation described in above is:
System: “Which are your hobbies?”
User: “Sport and music.”



Figure 4-12 Single answer in GUI



Figure 4-13 Multiple answer in GUI

(10) Confirmation answer

Definition. It specifies whether the user’s response is followed or not by an extra confirmation question.

Rationale. Like humans, systems that attempt to understand user’s input make mistakes. However, humans avoid misunderstandings by confirming doubtful input. MM systems have historically been designed so that they either request confirmation or not at all [McGe98]. If the system receives input that it finds uncertain, ambiguous or infeasible, or if its effect might be profound, risky, costly or irreversible, it may want to verify its interpretation of the command with the user. For instance, a system prepared to execute the command “Format hard disk” should give the user a chance to change or correct the command. Otherwise, the cost of such an error is task-dependent and can be immeasurable. [Blan06] argues that designers should use confirmations to ensure the correctness of a high risk irreversible input.

Therefore, the system should be able to request confirmation of the user’s command, as humans tend to do. Just like in human-to-human dialog, such confirmations are used to achieve common ground in HCI. Moreover, confirmations are an important way to reduce miscommunication. In fact, the more likely miscommunication is, the more frequently designers should introduce confirmations.

As MM UIs combine two or more interaction modalities, choosing the occurrence moment of the confirmation is another issue. Confirmation could occur for each modality or be delayed until the modalities have been fused. [McGe98] shows that confirmation after fusion reduces the time to perform manipulation tasks with the UI, making the interaction faster.

Values. Two values were identified:

- *With confirmation*: the system requests a confirmation of the previous answer from the user:

4. A Transformational Method for Producing Multimodal User Interfaces

- GUI: when deleting system files a confirmation question is recommended in order to verify the user's decision (Figure 4-14).
- VUI:
 - System: "Do you want to delete the file 'configsys.exe'?"
 - User: "Yes".
 - System: "Your answer was yes. Do you confirm?"
 - User: "Yes".
- *Without confirmation:* the system doesn't require any further confirmation:
 - GUI: the specification of age in a booking flight system is not a critical information and doesn't require any confirmation (Figure 4-15).
 - VUI:
 - System: "What is your age?"
 - User: "27".

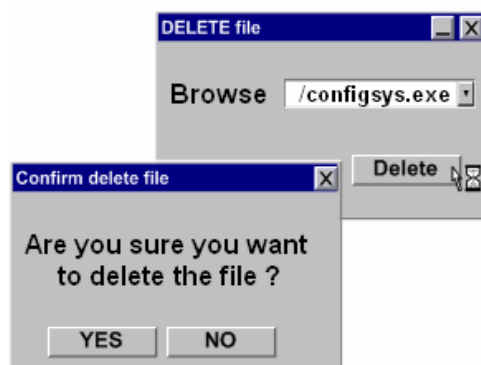


Figure 4-14 Confirmation message in GUI

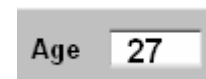


Figure 4-15 Non-confirmed message in GUI

(11) Answer order

Definition. It specifies the order in which the users can convey the answers.

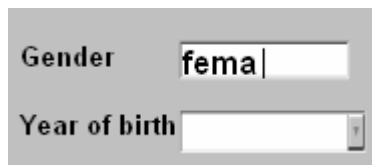
Rationale. [Pate97] introduces a formal specification of operators used to specify temporal relationships among tasks independent of the modality employed to achieve them. Among these operators we identify the enabling operator according to which one task enables a second one when it terminates. An order-independent operator is also introduced as a binary operator with the following semantic: at the beginning both tasks (T1 or T2) can be performed. However, as soon as the first action of task T1 (respective: T2) has been carried out, the whole task T1 (respective: T2) has to be performed before enabling the performance of task T2 (respective: T1). Such an operator is suitable when tasks T1 and T2 have to be sequentially executed, without imposing any restriction over which task to execute initially.

In VUIs many factors must be considered when designing if the order in which the information can be specified by the users is the same as the one requested by the system's prompts, but the most important thing is assessing the trade-off between flexibility and performance. The more designers constrain what user can say to an application, the less likely they are to encounter recognition errors [Java98]. On the other hand, allowing users to enter information flexibly can often speed the interaction (if recognition succeeds), feel more natural, and avoid forcing users to memorize commands.

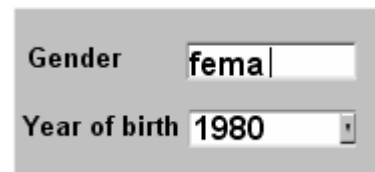
Values. Two design option values were identified:

4. A Transformational Method for Producing Multimodal User Interfaces

- *Order dependent*: the user has to convey the information in a predefined order:
 - GUI: in Figure 4-16 the user has to fill in first the gender after which the combobox for specifying the year of birth will be activated.
 - VUI: the user has to utter the information in a predefined order so as to be recognized by the system:
System: “*What are your gender and birth year?*”
User: “*I am male and I was born in 1980.*”
- *Order independent*: the user has the flexibility of conveying the information not necessarily considering the order in which it is requested by the system:
 - GUI: in Figure 4-17 the user can specify the gender and the year of birth in any order.
 - VUI: the user has the flexibility of uttering the answers in any order and the system is in charge of mapping the answers to the correct location:
System: “*What are your gender and birth year?*”
User: “*I was born in 1980 and I am male.*”
System: “*Your gender is male and you were born in 1980.*”



Gender fema |
Year of birth |



Gender fema |
Year of birth 1980 |

Figure 4-16 Order dependent answer in GUI Figure 4-17 Order independent answer in GUI

Hereafter we introduce a set of five design options that found their basis in the ergonomic criteria for the evaluation of human-computer interfaces presented in [Bast93]. After giving an insight into these criteria we abstracted a subset of them into design options. The selected criteria apply better with respect to the development process of ISs, while the rest of them refer to features of the resultant IS that can be checked only at the end of the development process and consequently they do not represent an interest for the current work. For instance, the *explicit user action* criterion refers to the relationship between the computer processing and the actions of the users. Accordingly, this relationship must be explicit, i.e. the computer must process only those actions requested by the users and only when requested to do so. But these actions can be identified only at run-time.

The subset of selected criteria is adapted for the development of MM UIs. Therefore, the resultant values specify the interaction type in which the design option can be concretized. Moreover, we associate to each value the corresponding CARE properties. The first design option concerns the input modalities, whereas the last four refer to different aspects of output modalities.

(12) Input

Definition. It specifies the modalities available to the user in order to provide information to the system.

Rationale. The nature of the task influences the input modality (modalities) selected by the users to perform them. Several researches demonstrated that tasks that are easy to perform in one modality may be difficult or even impossible to achieve using other modalities. For instance, [Meri06] shows that the vocal modality is mostly used in the

4. A Transformational Method for Producing Multimodal User Interfaces

case of a precise and short input data. [Lars06] provides a series of suggestions concerning the input modalities that are more appropriate depending on the nature of the tasks to achieve, the constraints of the physical device and the working environment of the user.

Studies made on a set of four basic manipulation tasks (i.e., select objects, enter text, enter symbols and enter sketches or illustrations) using three traditional input modalities (i.e., vocal (voice), graphical (keyboard and mouse) and tactile (stylus pen)) reveal the following observations: (1) object selection is easy with a pen or using voice to specify the desired object, but more difficult with the keyboard; (2) for entering the text all the modalities can be used but most users can speak and write easily; (3) entering mathematical equations, special characters and signatures is easy with a pen, awkward and time consuming with a mouse and most difficult with speech; (4) drawing simple illustrations and maps is easy with a pen, awkward with a mouse and nearly impossible with speech.

Some physical device constraints (e.g., size, shape, placement of the microphone, size of display, size of keys in a keypad) could also influence the input modality to be employed by the user. Therefore, the following suggestions are provided: (1) if user's hands are unavailable for use, then make speech available; (2) if user's eyes are busy or unavailable, then make speech available; (3) if the user is moving, then make speech available. In addition, users might work in environments that may not be ideal for some input modalities. The environment might be noisy or quite, light or dark, moving or stationary with a variety of distractions and possible dangers. Therefore, two additional suggestions are made: (4) if the user is in a noisy environment, then use graphical or tactile modality; (5) if the user's manual dexterity is impaired, then use speech.

We conclude that each input modality has its strengths and weakness. Consequently, a useful and efficient MM UI has to use the appropriate modality for each input.

Values. The possible values are:

- *Vocal (assignment).*
- *Graphical (assignment).*
- *Multimodal (equivalence, complementarity or redundancy).*

(13) Simple output

Definition. It specifies the modalities available to the system in order to produce information that will be further perceived by the user.

Rationale. In our thesis MM UIs consist of both vocal and graphical elements. There is no absolute need for a one-to-one mapping between them. Some information is better conveyed in vocal modality, other in graphical, but the majority of designers combine both modalities in most cases [IBM03b].

In general, welcome and introductory information can be well conveyed using voice to catch the user's attention as soon as the application starts. The elements containing brief information (i.e., short instructions) are well suited for voice. For instance, in an e-mail application that can read out loud the e-mail subject, users can individually choose to read or listen to a particular e-mail rather than browsing each one.

Some information is not easy to present using voice, such as graphics, diagrams and tables. These are better presented in visual format. If the designer wants to add voice to

4. A Transformational Method for Producing Multimodal User Interfaces

these visual elements, they need to give special consideration to the wording of the speech by emphasizing the key information it depicts. For example, if the designer wants to convey a pie chart using both graphical and vocal modalities, the application may say: “The biggest segment is ... and the smallest segment is ...” when the chart is displayed. Moreover, [Meri06] shows that the graphical modality is usually preferred by end-users while visualizing the output data as they have the possibility of re-reading them.

Since a MM application enables both vocal and graphical interaction, it is very important to keep a consistent *Sound, Look and Feel* of the application. To promote consistency, designers should use a consistent strategy for determining which information to present vocally, which to present graphically, and which to present using both modalities. Moreover, they should try to use the same terminology in both interfaces whenever possible. For example, the system shouldn't utter “Let's get started”, while the visual interface displays the “Welcome to the car rental system”. Synchronization is an issue specific to the MM environment. Since MM applications are conveying information to the user using both vocal and graphical modalities, they should be always synchronized [IBM03b]. The designer should avoid long paragraphs of information at one time because users may easily lose their attention while listening and reading. They should make paragraphs as brief as possible if they want to convey the information in both modalities. If they still want to convey the information using a long paragraph, it is better to convey it visually only so that users can choose to read it at their own speed.

Values. The possible values are:

- *Vocal (assignment).*
- *Graphical (assignment).*
- *Multimodal (equivalence, complementarity or redundancy).*

(14) Prompting

Definition. It specifies the modalities available to the system in order to lead the users to take specific actions whether it is data entry or other tasks.

Rationale. Good prompting guides the users and saves them from learning a series of commands [Bast93]. In addition, it allows them to know exactly the current modality, where they are in the dialogue as well as the actions that resulted in that context. Therefore, well designed prompts help users navigate in the application, reduce the errors and ensure a successful interaction with the application.

Deciding on an appropriate prompt with respect to the employed modality depends greatly on the content and context of the application [Java98]. If privacy is an issue, it is probably better not to have the computer speak out loud. On the other hand, even a little bit of spoken output can enable eyes-free interaction and can provide the user with the sense of having a conversational partner rather than speaking to an inanimate object.

The reflexive principle states that users tend to respond in the same manner and employ the same modality that they are prompted [Lars06]. Therefore, if the designers want to urge users to respond vocally they should use vocal prompts, whereas for graphical responses they should employ graphical prompts. In any case a MM prompt will provide users with the flexibility of selecting the appropriate modality to use.

Values. The possible values are:

- *Vocal (assignment).*

4. A Transformational Method for Producing Multimodal User Interfaces

- *Graphical (assignment).*
- *Multimodal (equivalence, complementarity or redundancy).*

(15) Immediate Feedback

Definition. It specifies the modalities available to the systems in order to provide an instantaneous reverse for user's input.

Rationale. The system should respond to each user action and its response should be explicitly conveyed in order to check the validity of the input [Cole85]. Feedback is also necessary for users to interpret the responses of the system to their actions. The feedback quality and rapidity are two important factors for the establishment of user's confidence and satisfaction as well as for the understanding of the dialog. These factors will allow them to gain a better understanding of the system's functioning. Therefore, the provided responses should be fast, with appropriate and consistent timing according to the considered input. The absence of feedback or a delayed feedback can be disconcerting to the user which may suspect a system failure and may undertake disruptive actions for the ongoing processes [Bast93].

Values. The possible values are:

- *Vocal (assignment).*
- *Graphical (assignment).*
- *Multimodal (equivalence, complementarity or redundancy).*

(16) Guidance

Definition. It specifies to the modalities available to the system in order to advise, orient, inform, instruct and guide the users throughout their interactions with the system thanks to UI elements such as messages, alarms, labels, icons.

Rationale. A good guidance facilitates learning and use of a system by allowing users to answer the following questions at any stage in the dialog [Cole85]: Where am I? (i.e., what dialog state?), What can I do? (i.e., what options are available?), How did I get here? (i.e., what sequence of actions brought me to this state?), Where can I go? (i.e., to what other dialog states can I progress?), How do I get there? (i.e., what control options are necessary to take me to the desired dialog state?). Ease of learning and ease of use that follow good guidance lead to better performances and fewer errors [Bast93].

Values. The guidance is sub-divided in:

- *Guidance for input:* any guidance offered to the user in order to guide him with the input. The possible values are:
 - *Vocal sub-divided into: Acoustic (assignment) and Speech (assignment).*
 - *Graphical sub-divided into: Textual (assignment) and Iconic (assignment).*
 - *Multimodal (equivalence, complementarity or redundancy).*For instance, a bell tone is an acoustic guidance which can be used to inform the user that the system is ready for the user's input.
- *Guidance for immediate feedback:* any guidance offered to the user in order to guide him/her with the feedback:
 - *Vocal sub-divided into: Acoustic (assignment) and Speech (assignment).*
 - *Graphical sub-divided into: Textual (assignment) and Iconic (assignment).*
 - *Multimodal (equivalence, complementarity or redundancy).*

4. A Transformational Method for Producing Multimodal User Interfaces

For instance, a percolating coffee pot is an acoustic guidance which can be used to inform the user that the application system is busy processing.

In order to exemplify the last five design option values, we consider in Figure 4-18 a design decision for a MM text input where the user has to provide his/her name [Stan06]. The value of the *prompt* design option is *multimodal* as the system indicates in a redundant manner the task to fulfill by employing two modalities: graphical modality (the label *Name*) and vocal modality used by the system to invite the user to input his name (1). The *guidance for input* has the type *iconic* and is composed of two elements (the microphone icon and the keyboard icon) indicating the available interaction modalities. User's *input* has the type *multimodal* as it can be provided in an equivalent manner by employing either the graphical modality (the user is typing his/her name in the text entry) or the vocal modality (the user is uttering his name using the microphone (2)). The *guidance for feedback* has the type *iconic* and is ensured by the loudspeaker icon, indicating the vocal feedback. The *immediate feedback* of the system following the user's input has the type *multimodal* as it is expressed by means of two redundant modalities: graphical (the result of users' typing) and vocal (the system is uttering the result of the input recognition (3)).

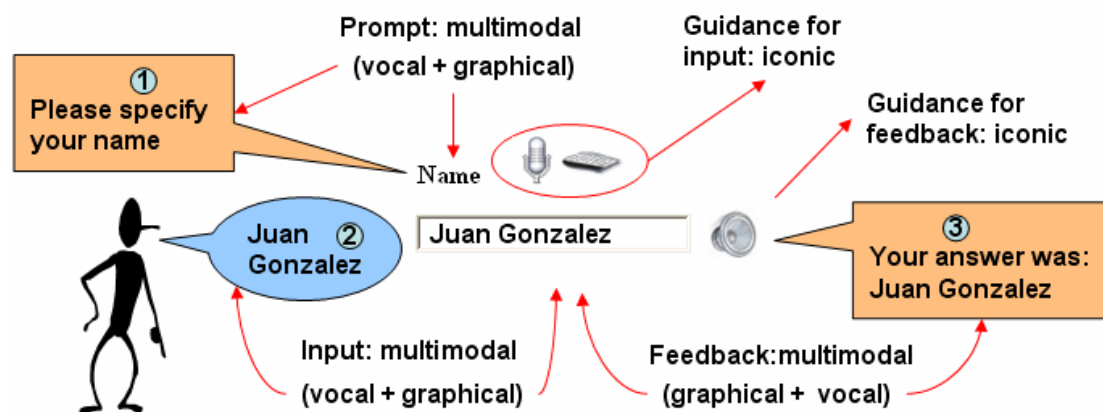


Figure 4-18 A possible design decision for a multimodal text input

4.2.4 Design space in the context of Design Rationale approach

Design rationale is an approach that supports several different alternative solutions for a given issue along with their justification and evaluation. The main purpose of this approach is to increase the quality of the designed ISs and their reusability for the development of future systems.

Several design rationale definitions were proposed in the literature, among which we adopted the one proposed in [Grub90]:

“A design rationale is an explanation of how and why an artifact, or some portion of it, is designed the way it is. A design rationale is a description of the reasoning justifying the resulting design - how structures achieve functions, why particular structures are chosen over alternatives, what behavior is expected under what operating conditions. In short, a design rationale explains the “why” of a design by describing what the artifact is, what it is supposed to do and how it got to be designed that way.”

4. A Transformational Method for Producing Multimodal User Interfaces

The approach appears in the context of ISs development where most of the existing methodologies (i.e., RUP, MERISE, OVID) suffer from a set of shortcomings [Laca05]:

- They do not allow to express the different explored design options, therefore it is impossible to know if the designers considered different options or not.
- They do not allow to justify the design option decisions, therefore the designer cannot argue their choices in a rational manner as the constraints that guided their decisions during the development life cycle are partially or totally forgot.
- They make it difficult to reuse the results of previous solutions even if several software engineering approaches (i.e., objects, components) tend to favour the code reusability from one project to another. Indeed, the source code remains the only reusable element whereas the other results obtained during the development process are usually non reusable for future design solutions.

As a response to the above identified shortcomings, design rationale provides several advantages [Laca05]:

- Allows to detect consistency and completeness issues early in the initial development phases [Conk88].
- Allows clarifying the reasons provided by the designers and forces them to argue their design decisions. Consequently, this will contribute to an increased quality of the final solution [Newm91].
- Forces designers to propose multiple solutions so that to enable the exploration of different potential results.
- Allows to provide a qualitative solution. For instance, if we consider that solution A is more suited than solutions B and C this does not mean that A is the best solution ever. The existing methodologies do not allow to assess the final solution and they provide it without being capable to state if it is a good, the best or the worst solution. This is the key difference between the current methodologies and the design rationale approach. At the end, both propose a solution but only the design rationale has the power to make explicit, justify and compare the final solutions with the non adopted ones, while still keeping track of the design decision history.

In order to visualise the dependencies among problems and their potential future solutions, a set of notations supporting design rationale approach are proposed in the literature. Among them we selected QOC (Question, Option, Criteria) [MacI91] due to its ease in generation and reading. It is a semi-formal notation represented as a diagram (Figure 4-19), decomposed in three columns (i.e., one for each element - questions, options, criteria) and the links between these elements. For each question (here, design option) we associate several options (here, design option values) that are further assigned to different criteria that favour (i.e., wide line) or not (i.e., dotted line) these options. The adopted options are emphasized in a rectangle. In QOC a question can be divided in sub-questions (e.g., question 2) in order to connect different diagrams. In addition, arguments can be assigned to support the evaluation of the links between the options and the criteria.

4. A Transformational Method for Producing Multimodal User Interfaces

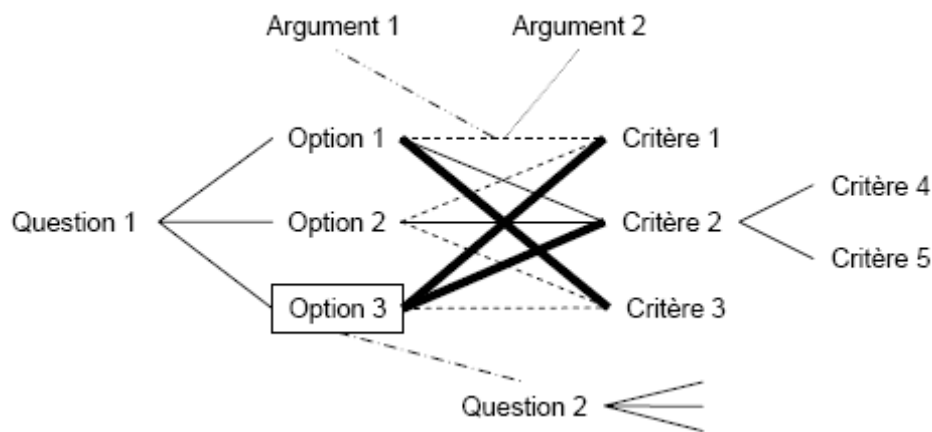


Figure 4-19 QOC diagram structure

Hereafter, we present the QOC representations by employing the Team tool [Laca05] for two design options composing our design space, whereas the rest of them are illustrated in Appendix D.

We are well aware of the fact that the considered criteria are somewhere subjective. However, we tried to decrease the subjectivity level by considering a set of proven properties [Gram96] to which any information system should adhere as well as a set of ergonomic criteria being experimentally assessed and successfully used to evaluate several types of UIs [Bast97]. The decision in the favour of one option or another is based on our previous experience with the design of UIs, but different options can be preferred if the set of criteria is modified (i.e., new criteria are added or the current ones are not considered important) depending on the context of use of the final solution.

Figure 4-20 presents the considered criteria for the *Sub-task guidance* design option. The *guided* option was selected due to its strong support for all the criteria. For instance, a good guidance facilitates learning the system and achieving the tasks by allowing users to be aware at any time which are the possible actions to perform as well as their consequences. Moreover, a good guidance leads to low number of errors and better retention over time. In contrast, the *unguided* option offers a weak support to the considered criteria.

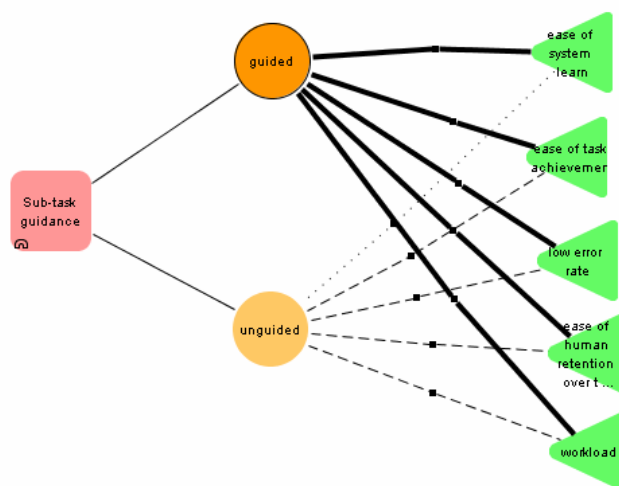


Figure 4-20 QOC representation of the sub-task guidance design option

Figure 4-21 illustrates the considered criteria for the *Confirmation answer* design option. The *with confirmation* option was selected due to its strong support for *error protection* and *error correction* criteria. Indeed, a system is much more robust if it prevents possible user errors such as accidental inputs and allows identifying them before validation rather than after. In addition, following error detection, users should be able to make corrections directly and immediately. However, the *without confirmation* option strongly supports a minimal number of actions to perform which results in fastest task achievement.

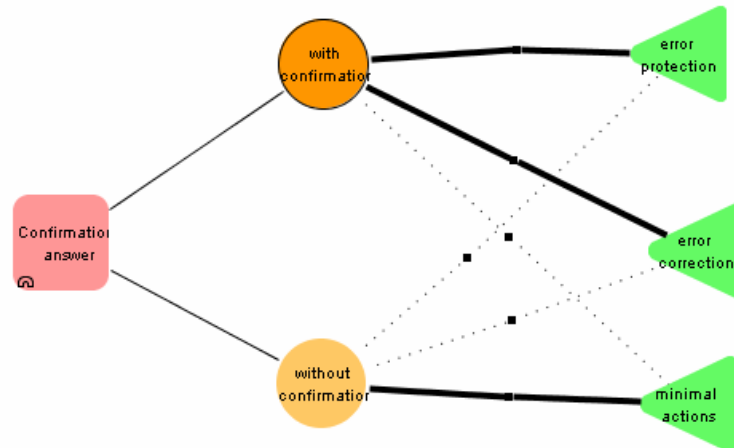


Figure 4-21 QOC representation of the sub-task guidance design option

4.3 Specification of transformations

4.3.1 Selection of model-to-model transformational approach

Model-to-model transformation approaches were the subject of several recent research works that tried to identify a mature foundation for specifying transformations between models [Varr02, Mell03, Agra03]. The high number of works on model-to-model transformation is mainly due to the Object Management Group (OMG) proposal on MDA [Mill03]. Several techniques have been surveyed in the literature [Czar03, Mens06], while the tools supporting them were analyzed in some works like [Medi07, Scha07]. Hereafter, we present the shortcomings of a couple of existing techniques identified in [Stan08]:

- *Imperative languages*: text-processing languages performing small text transformations (e.g., Perl, Awk) cannot be considered to specify complex transformation systems as they force the programmer to focus on very low-level syntactic details.
- *Relational approaches*: rely on declaration of mappings between source and target element type along with the conditions in which a mapping must be initiated. Relational approaches are generally implemented using a logic-based programming language and require a clear separation of the source and target models.
- *XSL Transformations*: is designed to specify transformations between different syntactical types of XML specifications. There are two main shortcomings of XSLT applied to achieve model-to-model transformations: (1) high complexity and lack of concision when managing complex sets of transformations rules and (2) lack of

4. A Transformational Method for Producing Multimodal User Interfaces

abstraction; progressively constructing the target XML specification entails an inclusion, in transformation rules, of syntactic details relative to target specification.

- *Common Warehouse Metamodel*: is an OMG specification that provides a set of concepts to describe model transformation grouped in transformation tasks, which are further grouped in transformation activities. A control flow of transformation can be defined between transformation tasks at this level. Even if transformations allow a fine-grained mapping between source and target elements, this specification does not provide us with a predefined language to specify the way elements are transformed one into another.

After identifying the shortcomings of the above transformational approaches we propose a transformational method based on *graph transformation rules* [Stan05] in order to progressively move from the uppermost level (i.e., the *Task and Domain Models*) to *Abstract Model* further refined into a more *Concrete Model* from which a *Final User Interface* is generated (Figure 4-22).

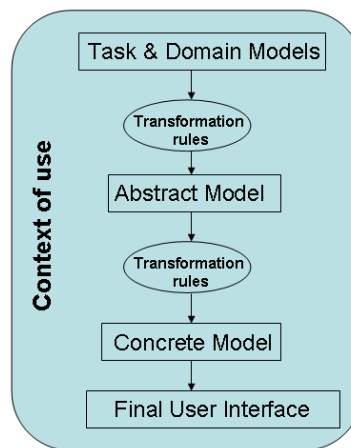


Figure 4-22 Progressive application of rule-based transformations

In the context of this research, we have selected the graph-based transformational approach. Our decision is motivated by [Czar03] which defines a taxonomy for the classification of several existing and proposed model transformation approaches. The taxonomy is described with a features model that makes explicit the different design choices for model transformations. Figure 4-23 traces the frontier of the features covered by the selected approach:

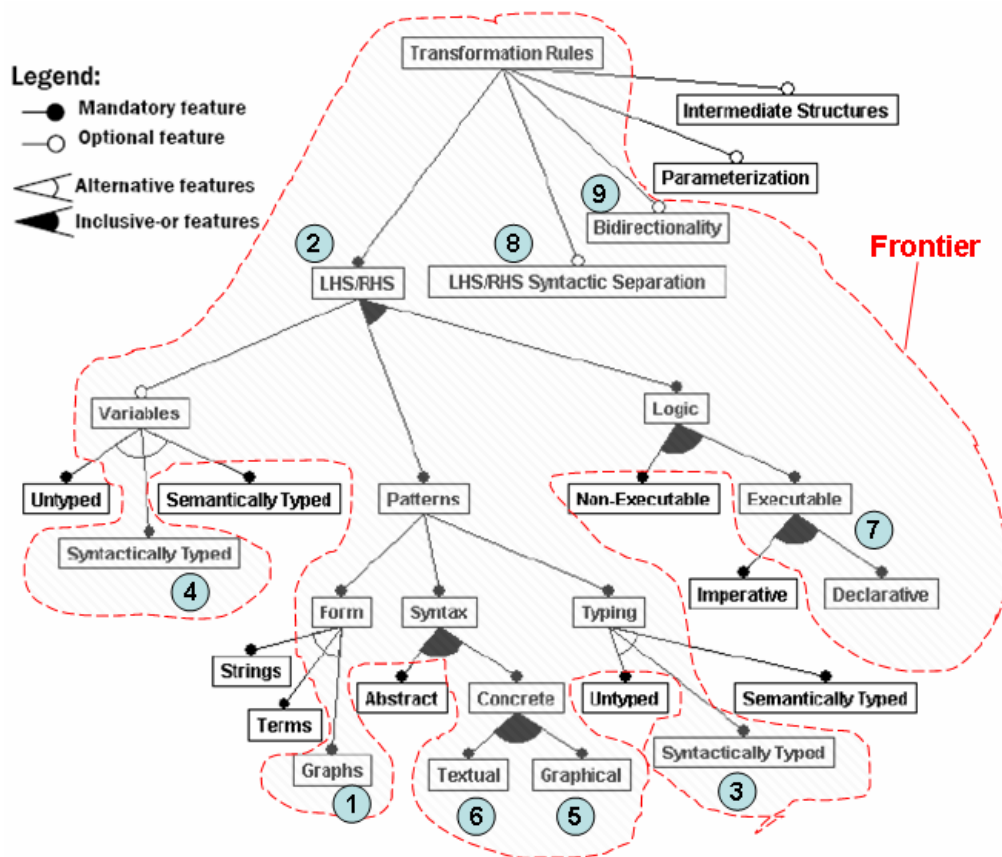


Figure 4-23 Identification of transformation rule approach features

- (1) **Graph-based patterns.** To ensure the progressive approach illustrated above, UsiXML provides a *Transformation Model* (Section 3.3.6) containing a set of rules that applies successive transformations to an initial representation. Transformations are encoded as *graph transformation rules* performed on the involved models expressed in their graph equivalent (Requirement 6. Ontological homogeneity). A set of *graph transformation rules*, known in the literature as *graph rewriting rules*, gathered along with the graph on which they apply (called *host graph*) define a *graph grammar*. The set of graph transformation rules are organized in a *transformation catalog* (Figure 4-24). The rules in a transformation catalog are structured in *development steps*. For instance, transforming a *Task Model* into an *Abstract Model* or an *Abstract Model* into a *Concrete Model* are two examples of development sub-steps. The development steps are further decomposed into *development sub-steps*. A development sub-step is realized by a unique *transformation system* and a transformation system is realized by a *set of graph transformation rules*.

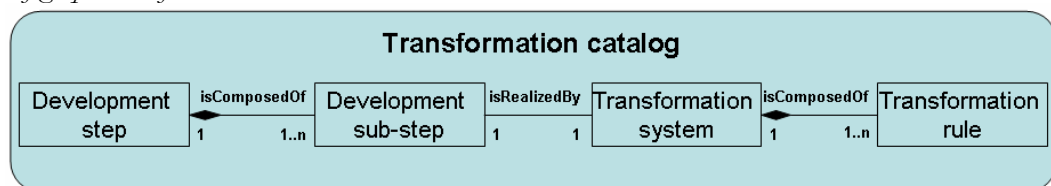


Figure 4-24 Structure of a transformation catalog

4. A Transformational Method for Producing Multimodal User Interfaces

- (2) **LHS/RHS.** The structure of the transformation rules identified in the taxonomy is composed of the couple (LHS, RHS) which ensures a pattern matching that selects a sub-graph in a graph structure and applies to this sub-graph any type of transformation (e.g., adding, deleting or modifying a node or an edge). Our approach considers conditional transformation rules so that a third graph (i.e., the NAC) is added to the initial structure. Thus, a transformation rule is defined by the graph triplet:

$$\text{Transformation Rule} = (\text{NAC}, \text{LHS}, \text{RHS})$$

where:

- *LHS (Left Hand Side)* of the rule: expresses a graph pattern that, if it matches the host graph, will be modified to result in another graph called *resultant graph*. A LHS may be seen as a condition under which a transformation rule is applicable.
- *RHS (Right Hand Side)* of the rule: is the graph that will replace the LHS in the host graph.
- *NAC (Negative Application Condition)* of the rule: expresses a pre-condition that have to hold false before trying to match LHS into the host graph. Several NACs may be associated to a rule.

Figure 4-25 illustrates how a transformation system is applied on G, where G is the graph representation of the initial UsiXML specification. The application of the rule implies several steps:

- (1) Find an occurrence (called *match*) of LHS into G. If several occurrences are identified, one of it is chooses non-deterministically.
- (2) Check that NAC does not match into G. If there is a match then skip to another occurrence of LHS.
- (3) Replace LHS by RHS.

G is consequently transformed into G' (the resultant UsiXML specification). All elements of G that are not covered by the match are left unchanged.

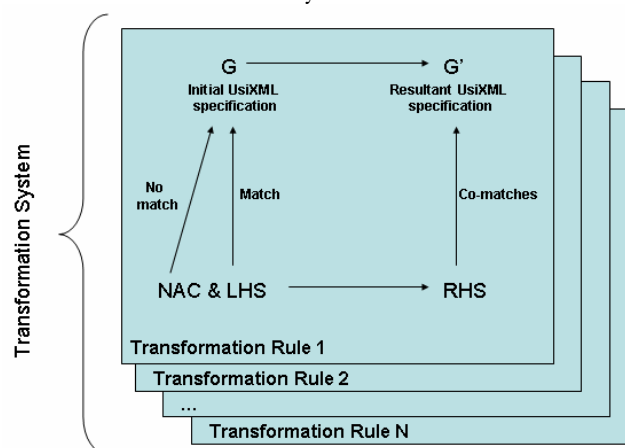


Figure 4-25 Characterization of transformation in UsiXML

- (3) **Syntactically Typed Patterns.** Represent patterns that are associated with meta-model elements whose instances it can hold. In our case, the typed graphs allow classifying nodes and edges by attaching types to them. Attaching several nodes (or

4. A Transformational Method for Producing Multimodal User Interfaces

edges) to the same types indicates a commonality in terms of properties between these nodes (or edges). Figure 4-26 illustrates the correspondence between, on one hand, node and edge types at the model level and, on the other hand, node and edge defined at the meta-model level.

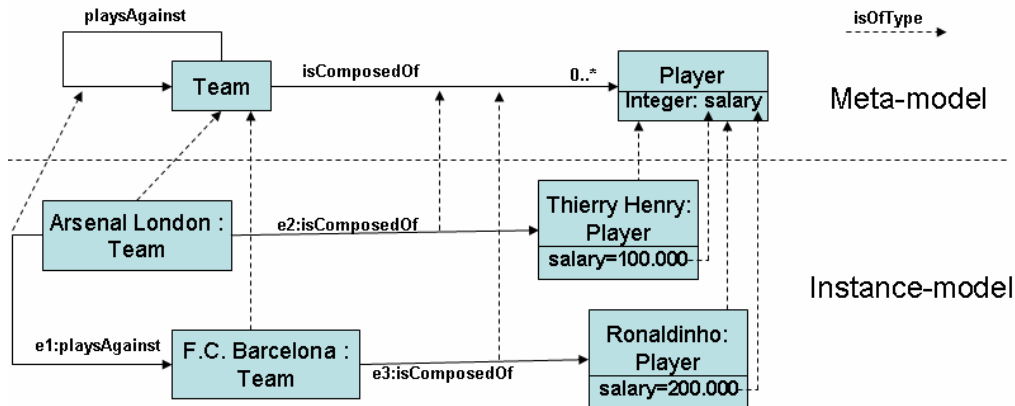


Figure 4-26 Syntactically typed patterns and variables

- (4) **Syntactically Typed Variables.** Similar to patterns, syntactically typed variables are variables that are associated with meta-model elements whose instances it can hold. Figure 4-26 shows the definition of the type of *salary* variable which is instantiated in the lower level with the values of the salaries for the two players.
- (5) **Graphical concrete syntax of the patterns.** The graphical concrete syntax of the transformation rules is based on the graphical formalism employed by Attributed Graph Grammar (AGG) environment, a generic tool for specifying and executing graph transformations [Ehri99]. Figure 4-27 illustrates the graphical notations for: nodes, edges, node and edge types and node and edge attribute values.

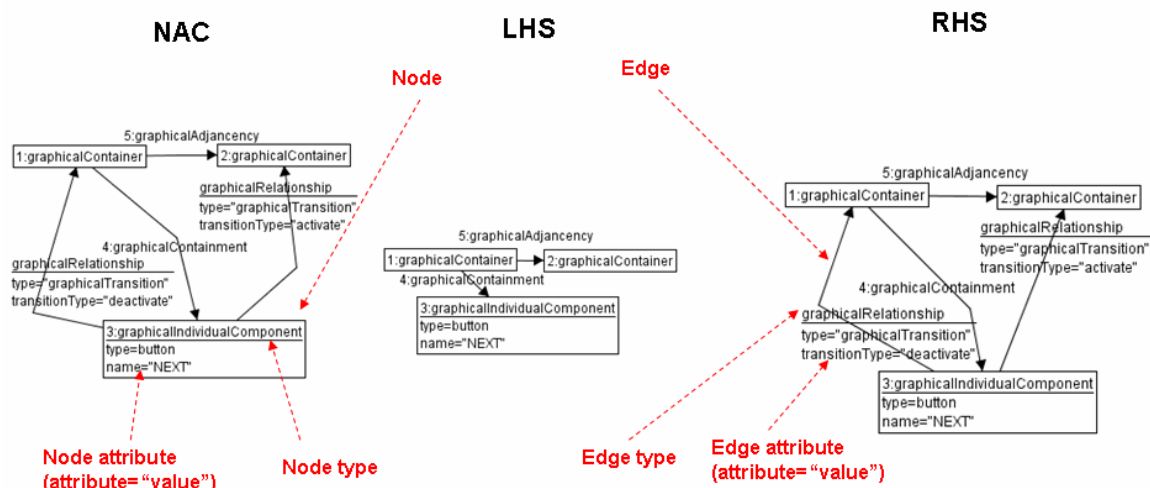


Figure 4-27 Graphical concrete syntax of the patterns

Figure 4-28 describes one of the rules employed in the transformation from the *Task Model* to the *Abstract Model*: for each task in the *Task Model* (see LHS) create an AIC in which it will be executed (see RHS) unless the task is not already executed into an AIC (NAC). In order to map the corresponding elements of the NAC, LHS and RHS of a

4. A Transformational Method for Producing Multimodal User Interfaces

rule, the graph formalism uses numbers in front of mapped nodes and edges (e.g., task 1 described in LHS corresponds to task 1 from NAC and RHS).

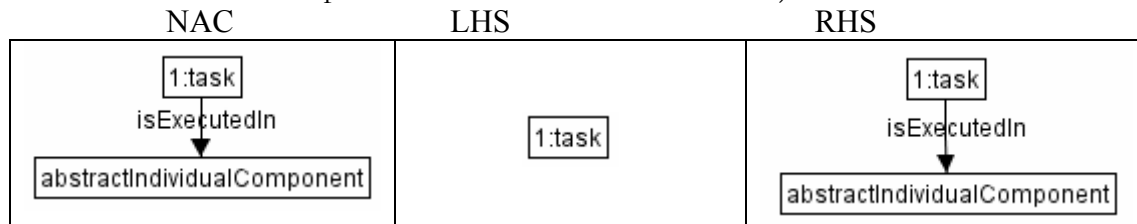


Figure 4-28 From Task Model to Abstract Model

- (6) **Textual concrete syntax of the patterns.** The textual concrete syntax of the rules is embedded in UsiXML. This textual syntax allows storing rules in an XML-based format. Figure 4-29 offers an example of the equivalent textual syntax of the rule illustrated in Figure 4-28.

```

<transformationRule id="Rule5-4" name="Rule1_from_task_to_abstract">
  <nac>
    <task ruleSpecificId="N1"/>
    <abstractIndividualComponent ruleSpecificId="N2"/>
    <isExecutedIn>
      <source sourceId="N1"/>
      <target targetId="N2"/>
    </isExecutedIn>
  </nac>
  <lhs>
    <task ruleSpecificId="L1"/>
  </lhs>
  <rhs>
    <task ruleSpecificId="R1"/>
    <abstractIndividualComponent ruleSpecificId="R2" name="x"/>
    <isExecutedIn>
      <source sourceId="R1"/>
      <target targetId="R2"/>
    </isExecutedIn>
  </rhs>
  <ruleMapping sourceId="L1" targetId="N1"/>
  <ruleMapping sourceId="L1" targetId="R1"/>
</transformationRule>

```

Figure 4-29 Textual syntax for expressing transformation rules

- (7) **Declarative executable logic.** Our graph grammars are based on formally defined execution semantics and have a declarative logic as they are described by graph patterns expressions.
- (8) **LHS/RHS Syntactic Separation.** Our implementation of the transformation rules makes clear distinction between the three components of a rule. Thus, the rule syntax (Figure 4-29) specifically marks the LHS, RHS and NAC elements.
- (9) **Bidirectionality.** Bidirectionality is achieved by defining two separate complementary unidirectional rules, one for each direction. [Limb04b] offers examples of forward and reverse engineering processes where transformation rules were designed to move forward and backward between different UI models.

4.3.2 Application strategy of transformation rules

The application strategy of the transformation rules is defined as the order in which they are applied to the initial graph [Limb04b]. This could be: concurrent, in an order independent manner or in a controlled sequential way. Two important issues have to be taken into consideration when deciding the application strategy: *confluence* and *termination*.

The *confluence* property refers to the ability of producing a unique resultant graph, thus raising the problem of the rule determinism. Parallel independent rules were shown to ensure the confluent property [Lowe93]. Moreover, the property can be proved intuitively if the transformation rules do not interfere one with each other, i.e. no rule deletes or introduces nodes that are needed by another one to match. But, in the current thesis the intrinsic nature of the process applied to an initial specification model determines us to we apply transformation rules that realize an incremental consolidation of it. In most cases, rules are inter-dependent as they rely on the information generated by the application of a previous rule. Therefore, in order to ensure the confluence property, we propose a special technique called Programmed Graph Rewriting [Schu97]. This technique uses graph rewriting rules as process units that may be composed arbitrarily using a set of pre-defined operators (e.g., sequences, parallel sequences, loops, tests) so as to obtain a desired algorithmic behaviour.

Our application strategy is presented in Figure 4-30. Once a development step is externally started, the first transformation system is executed. When it terminates, the second one is applied and so forth until the execution of the last transformation systems. Within the transformation system itself the transformation rules are applied following the same logic. The placement of the transformation rules is determined by the function played by each one in the corresponding development step, sub-step and transformation system.

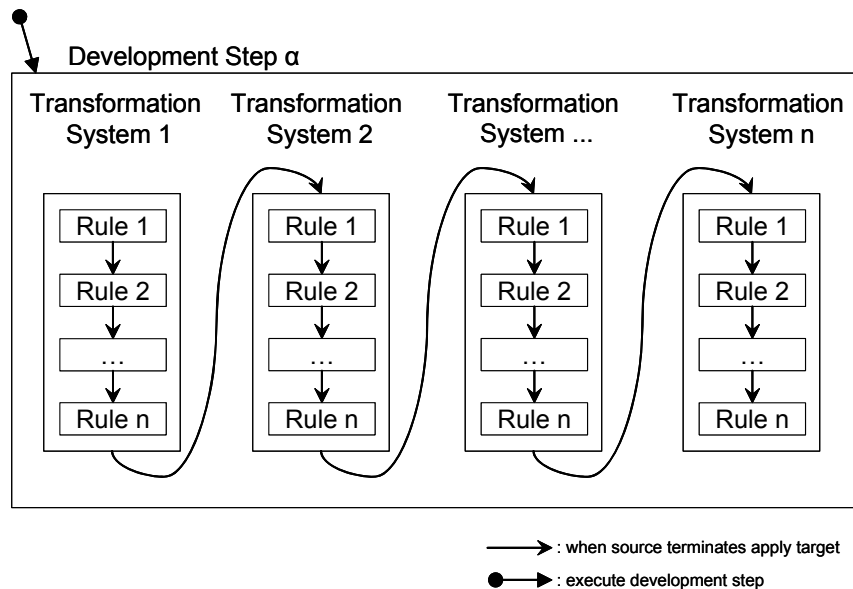


Figure 4-30 The application strategy of transformation rules

The *termination* property is satisfied if a transformation rules doesn't find any matches in the resultant graph. Consequently, a transformation system is terminated if each rule composing it is terminated. A development sub-step terminates if each of its

4. A Transformational Method for Producing Multimodal User Interfaces

transformation system terminates. A development step terminates if each of its sub-steps terminates.

Note that the problem of infinite looping may arise due to the non-deleting character of the rules. This issue is solved by replicating a part of the RHS in the NAC.

4.3.3 Shortcomings of the existing graph-based transformational approaches

By observing the current solutions that adopt a graph-based transformational approach for the model driven engineering of MM UI we have identified a list of shortcomings:

- Many transformation rules share some common parts either in the NAC, LHS or RHS and only slightly differ from one rule to another. Consequently, *many rules repeat common parts* without any connection between them and without factoring them out. Thus, many rules duplicate some significant portions of their NAC, LHS, and RHS.
- Due to this repetition, the transformation system that consists of the whole *set of transformation rules easily becomes huge and no longer scalable*. In addition, a static analysis of common portions of rules becomes a challenging task.
- The designer responsible for writing the rules to be fired by the transformation engine may only have limited means, formal or informal, to control the consistency of those rules that are similar, thus increasing *the risk for human error and redundancy*.
- *The scalability* of a transformation set for multi-target systems largely depends on its structure: if transformation rules are properly organized, then adding, removing or modifying a rule remains acceptable. But when this structure is poor, it is almost impossible to add new rules for another target without affecting significantly the rest of the rules in the same set.

4.3.4 Expanded model-to-model transformational approach

In the research literature the notion of *color* is used as a feature attached to tokens in High level Petri nets and used to distinguish between different data types carried throughout the network [Jens98]. In [Ehri99], the notion of color is currently defined at the level of type graph as a particular feature of the labels and enables to assign colors to nodes and edges. This imposes a set of restrictions as the color does not have any specific semantic meaning that allows manipulating and reasoning about graph transformations. Moreover, all nodes/edges of the same type must have the same color. Therefore, in order to cope with the shortcomings identified above, we expand the existing model-based approach by introducing the *color* as an explicit feature associated to the involved models that will add semantic to the transformation rules manipulating the elements of these models [Stan08]. The advantage of our contribution lies in the reusability, partially or entirely, of the transformation rules for developing UIs for target platforms that enable different interaction modalities than those previously available on the source platform.

4.3.4.a.1 Colored UsiXML concepts

The notion of color will make a distinction (Table 4-1) between the concepts corresponding to modality independent models (i.e., the Task, Domain, Mapping and

4. A Transformational Method for Producing Multimodal User Interfaces

Abstract Models) and those describing the modality dependent aspects (i.e., the Concrete Model):

- The concepts of the Task, Domain, Abstract and Mapping Models are represented in black. The selection was based on the analogy between the neutral character of the color and the neutral character of the above models with respect to the modality.
- The monomodal aspects of the Concrete Model consider a particular color for each modality: red for graphical modality and blue for vocal modality. Thus, the graphical concepts are represented in red, whereas the vocal concepts in blue. The relationships that reflect the monomodal aspects of the Concrete Model are said to be monocolored as they inherit their color from the common color of the source and target elements. The association of a particular color for each considered modality provides flexibility when extending the Concrete Model with concepts belonging to eventually newly introduced modalities as they can be associated to colors that haven't been used so far.
- The MM aspects of the Concrete Model consider the *cuiDialogControl* and *synchronization* relationships. These relationships are said to be multicolored as they inherit their color from the source element. For instance, a *cuiDialogControl* relationship that connects two graphical elements will be red, whereas its color becomes blue if the relationship connects two vocal elements. The *synchronization* relationship has associated the blue color as the source element is always a vocal element, but one can imagine the synchronization between an element belonging to a newly introduced modality (e.g., tactile) and a vocal element. In this case the color of the relationship will be the color associated to the new modality.

Concepts UsiXML Models			Elements	Relationships		Associated color
				Modality dependent	Modality independent	
Task			task	decomposition temporal	-	Black
Domain			domainClass	domainRelationship	-	Black
Abstract			AIO	abstractContainment abstractAdjacency cuiDialogControl	-	Black
Mapping			-	manipulates triggers updates isExecutedIn isReifiedBy	-	Black
Concrete	Monomodal	Graphical	GIO	graphicalContainment graphicalAdjacency graphicalTransition	-	Red

4. A Transformational Method for Producing Multimodal User Interfaces

		Vocal	VIO	vocalContainmemnt vocalAdjacency vocalTransition	-	Blue
	Multimodal aspects		-	-	cuiDialogControl synchronization	The relationship inherits the color of the source object

Table 4-1 Color associated to the UsiXML model concepts

4.3.4.a.2 Colored graphs

In [Limb04b] the graph structure (see Definition1) used as an abstract syntax for defining the underlying formalism of a model-to-model transformational approach is progressively consolidated into a single graph category called (Identified, Labeled, Constrained, Typed)-Graph.

Definition 1. A graph g is defined by a quadruple $(V, E, source_g, target_g)$ such that:

1. V is a finite set of vertices
2. E is a finite set of edges
3. $source_g: E \rightarrow V$, is an injective function that assigns a source vertex to every edge from E ;
4. $target_g: E \rightarrow V$, is an injective function that assigns a target vertex to every edge from E .

Hereafter we extend this category with the concept of colored graph (Definition 2), as a graph in which a color is assigned to all its components.

Definition 2. Let $COL = (NodeColor, EdgeColor)$ be a pair of disjoint and finite sets of predefined colors. g is said to be a (COL) -Graph iff g is a pair (g, Col) such that:

1. g is a graph (see definition 1)
2. Col is a pair of total functions attaching a color to each node and edge of the graph: $Col = (Col_v, Col_e)$, where $Col_v: V \rightarrow NodeColor$ and $Col_e: E \rightarrow EdgeColor$

Depending on the level of abstraction on which it is defined, the properties of these functions are different. If the graph structure is exploited to describe the model level (Table 4-1), the color functions (i.e., Col_v and Col_e) are surjective (i.e., each color is assigned to a graph component). If the graph structure is exploited to describe the instance level then different graph components may share the same color. Depending on the number of non-neutral color (i.e., different of black color) with respect to the interaction modality, the (COL) -Graph can be specialized into:

- *Monocolored* (Definition 3): the graph has at most one color in the codomain of Col_v that is different of the black color. This implies that the cardinality of the image of Col_e could be: 0 if the graph has a single vertex, 1 if the edge describes the mapping relationship between an abstract and a concrete element, or 2 if the mapping applies over two concrete elements.

Definition 3. g is said to be a (MONOCOL)-Graph iff:

1. g is a (COL)-Graph
2. $1 \leq |\text{Im}(\text{Col}_v)| \leq 2$
3. $0 \leq |\text{Im}(\text{Col}_e)| \leq 2$
4. $\exists ! c \in \text{NodeColor} \setminus \{\text{black}\}$

- *Multicolored* (Definition4): the graph has at least two colors in the codomain of Col_v that are different one of each other and different of the black color.

Definition 4. g is said to be a (MULTICOL)-Graph iff:

1. g is a (COL)-Graph
2. $|\text{Im}(\text{Col}_v)| \geq 2$
3. $\exists c_1, c_2 \in \text{NodeColor} \setminus \{\text{black}\} \mid c_1 \neq c_2$

The graph category identified in [Limb04b] as (I, L, C, TY)-Graph is consolidated with the newly introduced feature into a new single graph category called (Identified, Labeled, Constrained, Typed, Colored)-Graph, in short (I, L, C, TY, COL)-Graph.

Definition 5. g is an **(Identified, Labeled, Constrained, Typed, Colored)-Graph** iff:

1. g is a graph (Definition 1)
2. g is an identified graph
3. g is a labeled graph
4. g is a constrained graph
5. g is a typed graph
6. g is a colored graph (Definition 2)

The advantage of this new consolidation relies on its modularity feature which allows to form other graph categories by combining the features “a la carte”.

4.3.4.a.3 Operations over colored graphs

The previously introduced notions allow us to define two operations over colored graphs:

- *Merging operation* (Definition 6): a (MULTICOL)-Graph results by merging two (COL)-Graphs (g, h). The color functions (Col_g and Col_h) of these graphs are restrictions of the colored functions ($\text{Col}_{v(r)}, \text{Col}_{e(r)}$) of the resultant graph r to the domain values of the initial graphs, respectively.

Definition 6. Let g and h be two (COL)-Graphs defined by $(V_g, E_g, \text{source}_g, \text{target}_g)$ and $(V_h, E_h, \text{source}_h, \text{target}_h)$, respectively. The result of the *merging operation* defined between g and h ($g \boxplus h = r$) is a graph r , where:

1. r is a (MULTICOL)-Graph
2. $\text{Col}_{v(r)}: V_g \cup V_h \rightarrow \text{NodeColor}_g \cup \text{NodeColor}_h$,
 $\text{Col}_{e(r)}: E_g \cup E_h \rightarrow \text{EdgeColor}_g \cup \text{EdgeColor}_h$,
 $\text{Col}_{v(r)}|_{V_g}(v) = \text{Col}_{v(g)}(v)$ $\text{Col}_{v(r)}|_{V_h}(v) = \text{Col}_{v(h)}(v)$
 $\text{Col}_{e(r)}|_{E_g}(e) = \text{Col}_{e(g)}(e)$ $\text{Col}_{e(r)}|_{E_h}(e) = \text{Col}_{e(h)}(e)$

- *Splitting operation* (Definition 7): a (MONOCOL)-Graph g results by splitting a (MULTICOL)-Graph r upon one color from the set of vertices colors different

4. A Transformational Method for Producing Multimodal User Interfaces

from *black*. The color functions ($\text{Col}_{v(g)}$, $\text{Col}_{e(g)}$) of the resultant graph are restrictions of the colored functions ($\text{Col}_{v(r)}$ and $\text{Col}_{e(r)}$) to the domain values of the initial graph, respectively.

Definition 7. Let r be a (MULTICOL)-Graph defined by $(V_r, E_r, \text{source}_r, \text{target}_r)$ and c a color where $c \in \text{NodeColor}_r \setminus \{\text{black}\}$. The result of the **splitting operation** of the graph r upon the color c ($r_{[c]} = g$) is a graph g defined by $(V_g, E_g, \text{source}_g, \text{target}_g)$, where:

1. g is an (MONOCOL)-Graph, with:

$$\text{NodeColor}_g = \{c\} \cup \{\text{black}\} \cap \text{NodeColor}_r$$

$$\text{EdgeColor}_g = \{c\} \cup \{\text{black}\} \cap \text{EdgeColor}_r$$
2. $V_g = \{v \mid \text{Col}_g(v) \in \text{NodeColor}_g\}$
 $E_g = \{e \mid \text{Col}_g(e) \in \text{EdgeColor}_g\}$
 $\text{source}_g(e) = \text{source}_r|_{E_g}(e)$, $\text{target}_g(e) = \text{target}_r|_{E_g}(e)$
3. $\text{Col}_{v(g)}: V_g \rightarrow \text{NodeColor}_g$, $\text{Col}_{e(g)}: E_g \rightarrow \text{EdgeColor}_g$
 $\text{Col}_{v(g)}(v) = \text{Col}_{v(r)}|_{V_g}(v)$ and $\text{Col}_{e(g)}(e) = \text{Col}_{e(r)}|_{E_g}(e)$

4.3.4.a.4 Colored transformation rules

The integration of the color as a new graph feature of our graph-based transformational approach allows the introduction of the notion of *colored transformation rule* ((COL)-TR), which can be of two types:

- *Monocolored transformation rule* (Definition 8): is a transformation rule in which at least one of the components of the rule is a (MONOCOL)-Graph.

Definition 8. Let TR be a transformation rule, with $\text{TR} = (\text{NAC}, \text{LHS}, \text{RHS})$. TR is said to be (MONOCOL)-TR iff $\exists g \in \{\text{NAC}, \text{LHS}, \text{RHS}\}$, where g is a (MONOCOL)-Graph.

The monocolored transformation rules are employed in the generation of monomodal UIs. The colors are given by the colors of the concrete concepts involved in the transformation rule (Table 4-1).

- *Multicolored transformation rule* (Definition 9): is a transformation rule in which at least one of the components of the rule is a (MULTICOL)-Graph.

Definition 9. Let TR be a transformation rule, with $\text{TR} = (\text{NAC}, \text{LHS}, \text{RHS})$. TR is said to be (MULTICOL)-TR iff $\exists g \in \{\text{NAC}, \text{LHS}, \text{RHS}\}$, where g is a (MULTICOL)-Graph.

The multicolored transformation rules are employed in the generation of MM UIs. The colors are given by the colors of the concrete concepts involved in the transformation rule (Table 4-1).

By analogy with the merging and splitting operations specified over graphs, we define hereafter the same operations over transformation rules.

- *Merging* two or more different colored transformation rules enables to generate multicolor rules (Definition 10). This operation is the cornerstone of the factoring out activity.

Definition 10. Let TR_1 and TR_2 be two (COL)-TRs, with $\text{TR}_1 = (\text{NAC}_1, \text{LHS}_1, \text{RHS}_1)$ and $\text{TR}_2 = (\text{NAC}_2, \text{LHS}_2, \text{RHS}_2)$. The result of the **merging operation** defined between TR_1 and TR_2 ($\text{TR}_1 \boxplus \text{TR}_2 = \text{TR}_3$) is a transformation rule $\text{TR}_3 = (\text{NAC}_3, \text{LHS}_3, \text{RHS}_3)$, where:

1. TR_3 is a (MULTICOL)-TR
2. $NAC_3 = NAC_1 \boxed{M} NAC_2$
3. $LHS_3 = LHS_1 \boxed{M} LHS_2$
4. $RHS_3 = RHS_1 \boxed{M} RHS_2$

If NAC_1 and NAC_2 share a common black element, they are merged in order to generate the NAC_3 of the resultant rule. If not, the two NACs will be aggregated in the resultant rule giving rise to two NACs. Splitting a multicolored transformation rule upon one color enables the designer to generate a monocolored rule.

- *Splitting* a multicolored transformation rule (Definition 11) upon a color enables the designer to generate a monocolored transformation rule.

- Definition 11.** Let $TR_1 = (NAC_1, LHS_1, RHS_1)$ be a (MULTICOL)-TR and $c \in \{\text{NodeColor}_{NAC} \cup \text{NodeColor}_{LHS} \cup \text{NodeColor}_{RHS}\} \setminus \{\text{black}\}$. The result of the splitting operation of the transformation rule TR_1 upon the color c ($TR_{1[c]} = TR_2$) is a transformation rule $TR_2 = (NAC_2, LHS_2, RHS_2)$, where:
1. TR_2 is a (MONOCOL)-TR
 2. $NAC_2 = NAC_1 [c]$
 3. $LHS_2 = LHS_1 [c]$
 4. $RHS_2 = RHS_1 [c]$

As a result of the Definitions 10 and 11, we reached the following conclusion: *A multicolored transformation rule is the result of the merging operation applied over all its splittings upon each non-neutral color of the nodes.*

4.3.4.a.5 Colored transformation rules at a glance

Thanks to the introduction of colors, the total amount of rules to be specified by the designer is significantly reduced. For a particular widget of a UI involving two interaction modalities (e.g., graphical and vocal), two monocolored rules had to be applied so far. These two rules can now be merged into a single multicolored rule that can be treated as follows: (1) if the designer needs to ensure both interaction modalities the multicolored rule has to be applied, (2) if the designer needs to ensure only one type of interaction (i.e., graphical or vocal) the rule has to be split upon the color assigned to the considered interaction. The flexibility of the colored rules is illustrated hereafter based on two examples that show its benefits. The first set of transformation rules are used to generate graphical and/or vocal containers. Figure 4-28 (a) presents the monocolored rule that is the result of the splitting operation applied over the rule in Figure 4-28 (c) upon the red color. It generates *groupBox* elements that embed an *outputText* (i.e., a label) and an *image-Component* guiding the user with the available interactions to use (i.e., mouse and keyboard). If the designer wants to ensure the same functionality but enabling just the vocal interaction, the rule illustrated in Figure 4-28 (b) has to be executed. It is the result of the splitting operation applied over the rule in Figure 4-28 (c) upon the blue color and is used to generate *vocalGroup* elements. On the other hand if the designer wants to ensure a MM interaction the rules in Figure 4-28 (a) and (b) have to be merged. The resultant rule is illustrated in Figure 4-28 (c) and generates both *groupBox* and *vocalMenu* elements.

4. A Transformational Method for Producing Multimodal User Interfaces

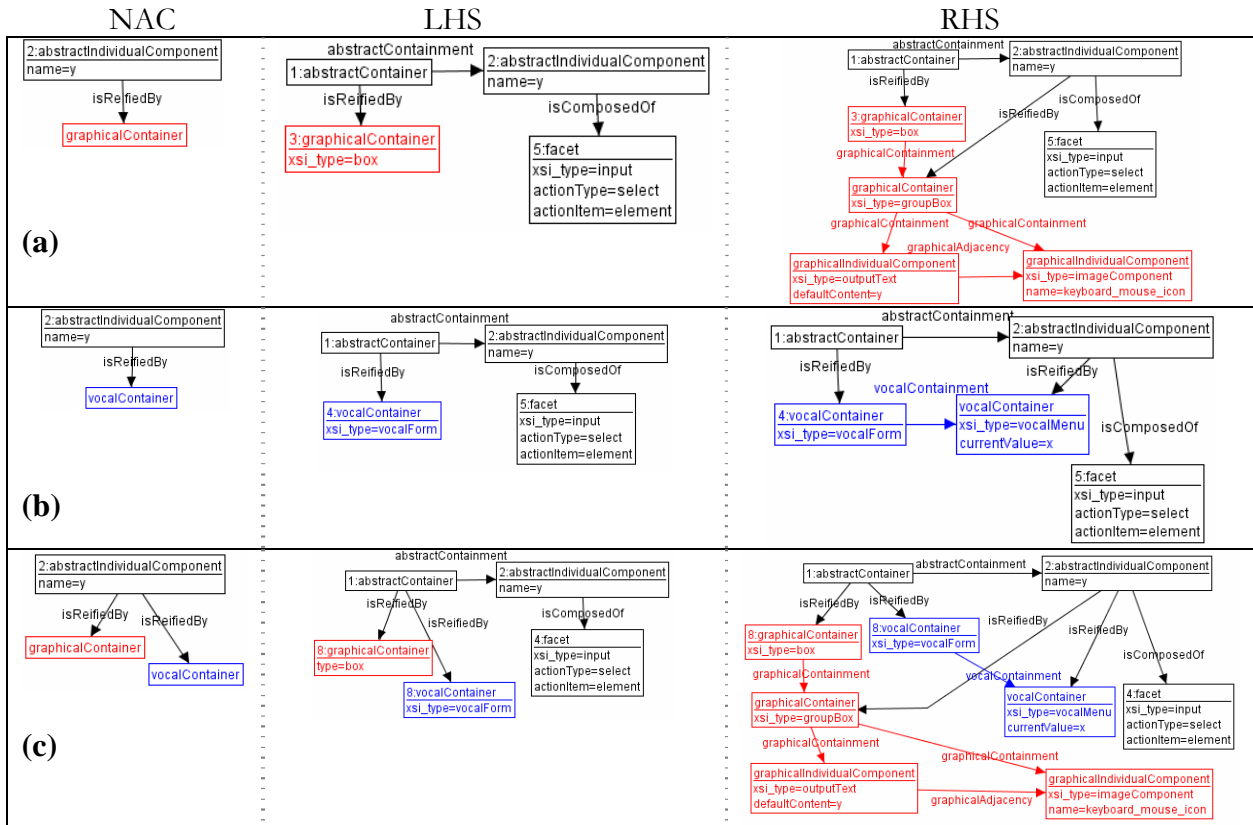


Figure 4-31 Monocolored transformation rule generating: (a) groupBox elements; (b) vocalMenu elements; (c) groupBoxes and vocalMenu elements

4.3.4.a.6 Benefits of colored transformation rules

We consider that our contribution could be applied in any area where factorization could be a solution for rules with a significant portion of the NAC, LHS or RHS that is duplicated. Therefore, the following benefits could be obtained:

- *Reduced number of rules* to be specified and applied: thanks to the introduction of the colors, each pair of graphical/vocal rules can be merged into a single one, reducing thus the number of rules to apply to half. In addition, as more interaction types are considered, the more benefit will be gained thanks to the multicolored transformation rules
- *High scalability*: if the need for a new modality arise (e.g., tactile modality) a new monocolored transformation rule that is duplicating the common part of its modality counterparts rules (i.e., the abstract elements represented in black) had to be developed. Thanks to the colored transformation rules, the development of a new rule, and thus of the duplicating elements, is avoided. A simple integration of the new concepts assigned to the introduced modality and their mapping to the abstract elements in the already existing multicolored rule is sufficient to achieve a direct modification. As a result, a new multicolored rule is obtained which can be applied in the generation of MM UIs considering graphical vocal and tactile interactions.

4.3.5 Transformation rule catalog

4.3.5.a Structure of the transformation rule catalog

In the current dissertation, the introduced design space is supported by a transformational approach that applies transformational rules over the involved models. Based on a theoretical analysis of the development sub-steps previously illustrated and due to their great number, the transformation rules were gathered in a Transformation Rule Catalog (Appendix B) in order to offer a complete and systematic arrangement with the following structure:

- I. *Transformation rules that support design options:* for each design option value we provide the rule that supports the generation of the abstract elements. Further we provide the multicolored rule that concretizes these elements into graphical and vocal elements. Depending on the type of interaction to ensure, the multicolored rule can be directly applied (i.e., for MM interaction) or splitted upon the color assigned to the desired modality
- II. *Additional transformation rules:* is composed of a set of transformation rules that provides supplementary support for the sub-steps that are not covered, totally or partially by the design space (e.g., Sub-step: Transformation rules for the selection of AICs).

4.3.5.b Design space coverage

Based on the draw up Transformation Rule Catalog we identify hereafter the mappings between each design value and the transformation rule(s) supporting it. Thus, Figures 4-29 to 4-35 specify the corresponding rule number in the catalog (i.e., R_i). Figure 4-35 describes the set of design options for which a stylistics was not required and considers the following notations: V=Vocal, G=Graphical, MM=Multimodal, T=Textual, I=Iconic, A=Acoustic, S=Speech.

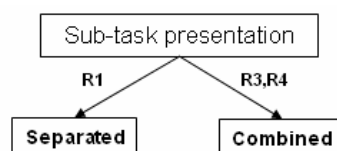


Figure 4-32 Transformation rules supporting sub-task presentation

4. A Transformational Method for Producing Multimodal User Interfaces

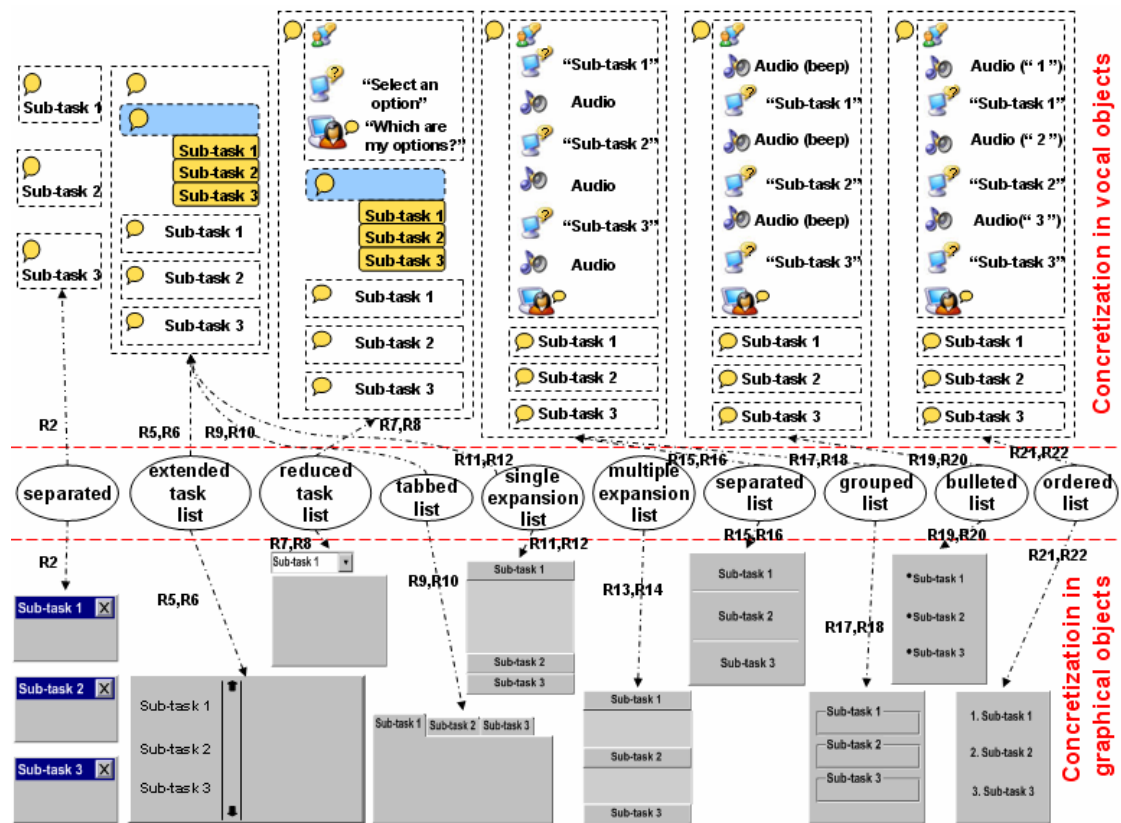


Figure 4-33 Transformation rules supporting the vocal and graphical concretization of sub-task presentation values

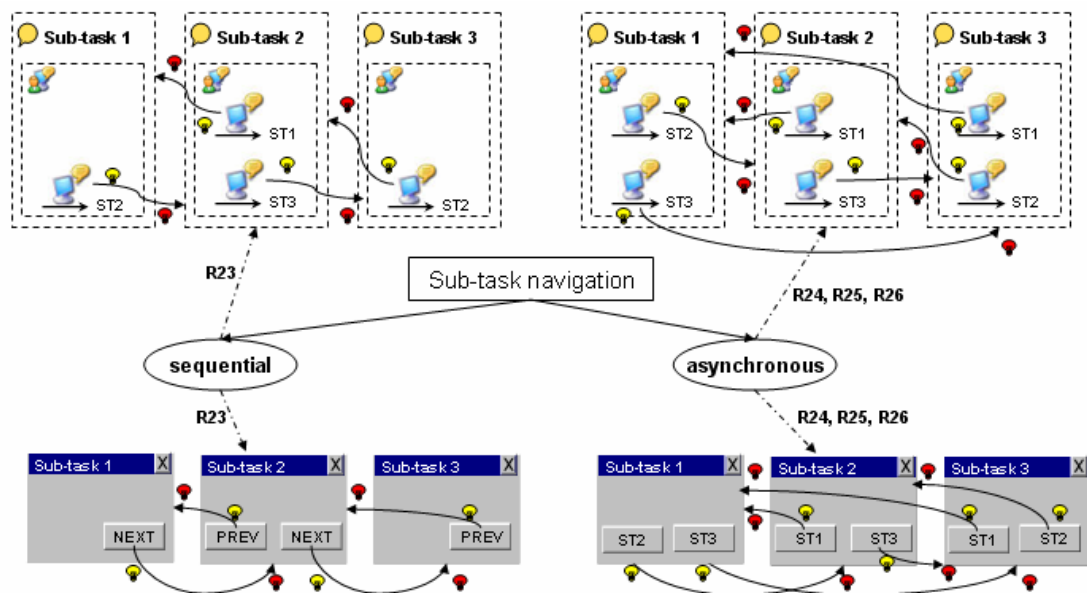


Figure 4-34 Transformation rules supporting sub-task navigation values for graphical and vocal concretization

4. A Transformational Method for Producing Multimodal User Interfaces

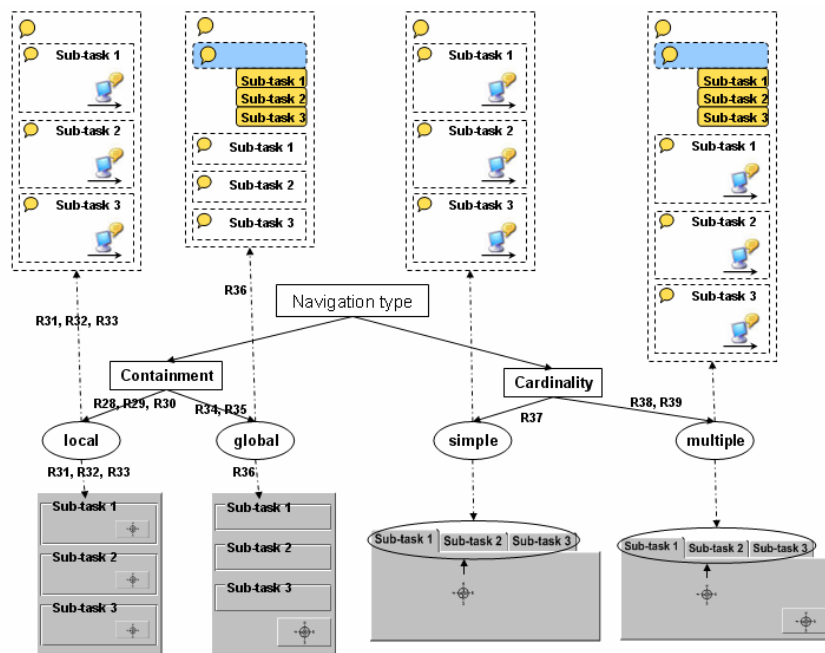


Figure 4-35 Transformation rules supporting navigation type values for graphical and vocal concretization

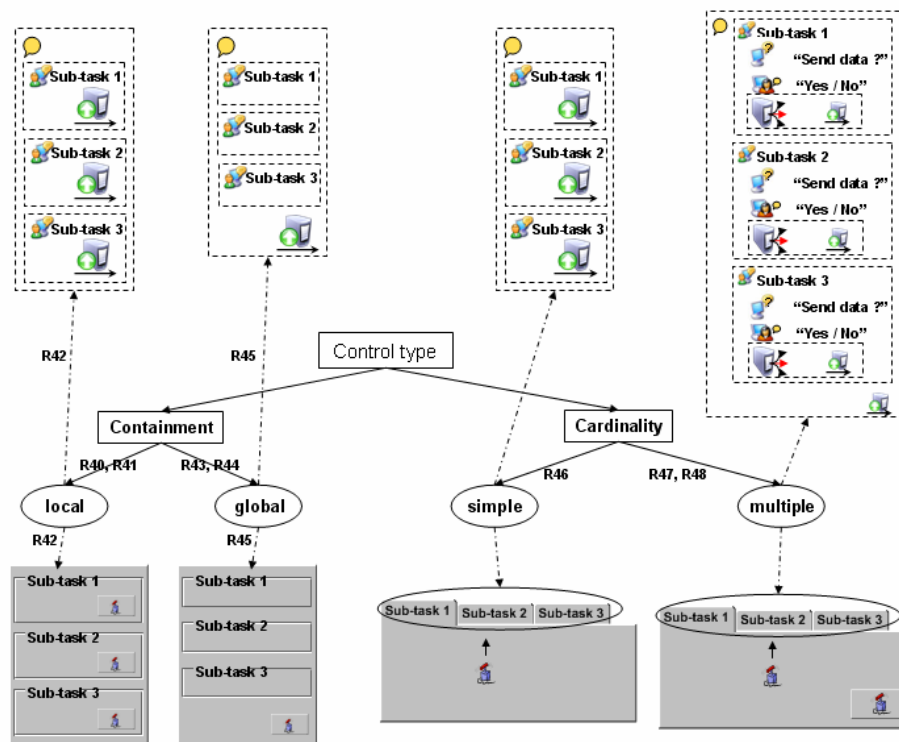


Figure 4-36 Transformation rules supporting control type values for graphical and vocal concretization

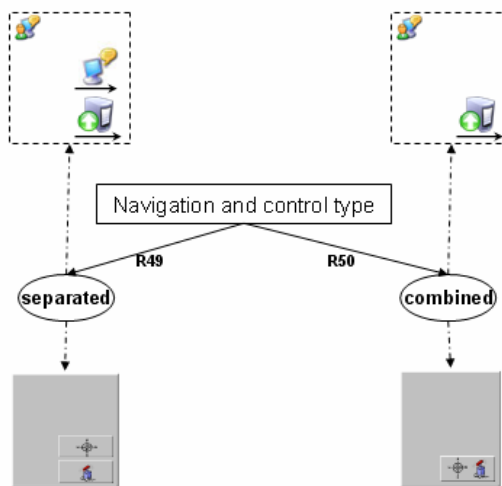


Figure 4-37 Transformation rules supporting navigation and control type values for graphical and vocal concretization

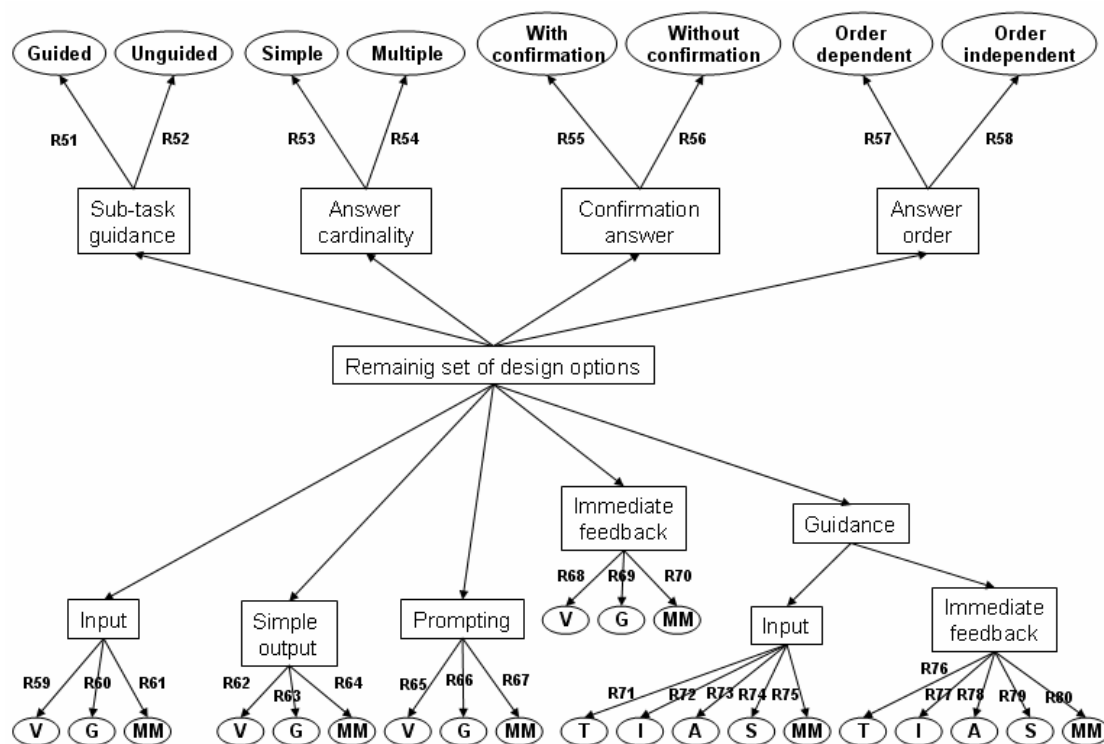


Figure 4-38 Transformation rules supporting the remaining set of design options for which a stylistics was not assigned

4.4 The four steps of the transformational approach

Our transformational approach involves a method which consists of a forward engineering process composed of four steps [Stan05] illustrated in Figure 4-39.

4. A Transformational Method for Producing Multimodal User Interfaces

1. *Step 1: Construct the Task and Domain Models:* the task and domain models are specified first so as to initiate the forward engineering process.
2. *Step 2: From Task and Domain Models to Abstract User Interface Model:* consists of producing one or many AUIs from the previously specified models.
3. *Step 3: From Abstract User Interface Model to Concrete User Interface Model:* from each AUI Model obtained in the previous step, different CUIs Models specifying graphical, vocal and MM UIs are derived.
4. *Step 4: From Concrete User Interface Model to Final User Interface:* from each CUI, a corresponding FUI can be produced by automated model-to-code generation. Thus, for GUIs we generate XHTML code, for vocal UIs we produce VoiceXML code, while MM UIs are specified using X+V language.

The approach is not addressing only the incremental aspects of the development process where the FUI is reached starting from a Task and Domain Models that are sequentially reified into more concrete models. It also supports an iterative approach related to any software development process, where initial requirements are continuously updated according to end-user requests. Therefore, two situations can be encountered:

- If the starting point of our process (i.e., the Task and Domain models) requires updates, then transformation rules can be performed over these models so that to respond to the requested requirements
- If an intermediate model (e.g., the Abstract Model) requires updates, then transformation rules can be performed over the model itself and over the upper models so that to ensure consistency with the requested requirements.

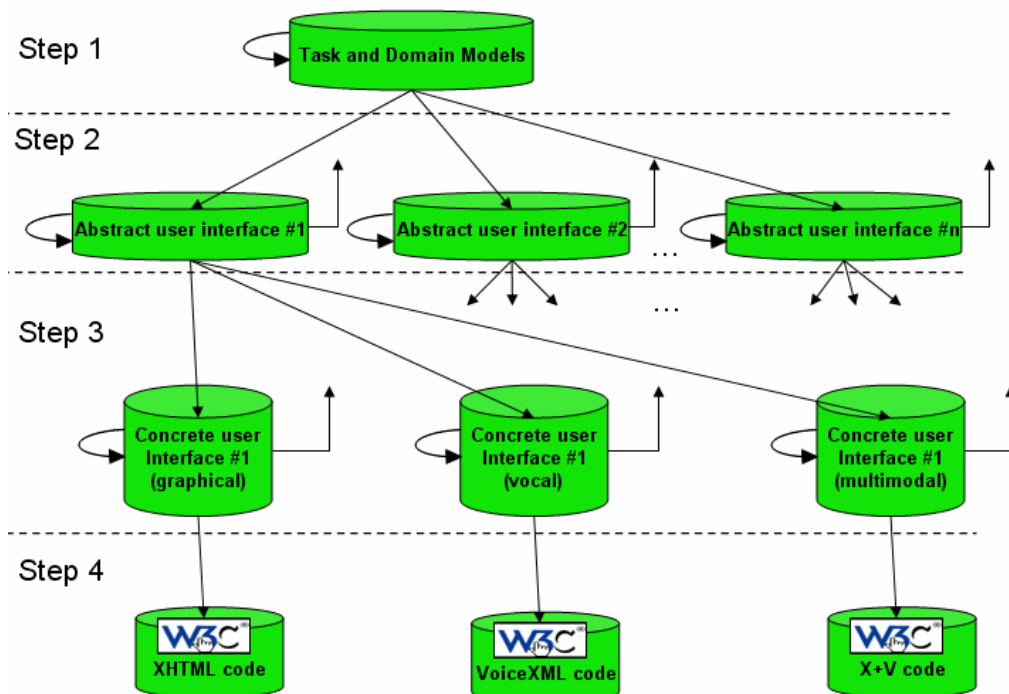


Figure 4-39 General development scenario of UI

In [Limb04b] steps 2 and 3 are further decomposed into sub-steps which consist of transformation systems applied in order to generate GUIs. The VUIs are also addressed but in a lower degree. The current thesis defines transformation systems by associating

4. A Transformational Method for Producing Multimodal User Interfaces

the design options defined in Section 4.2 to the different identified sub-steps of the transformational process (Figure 4-40). Thanks to the extended ontology of models described in [USIX07] we add more focus on vocal UIs and expand the coverage area to MM UIs, as well.

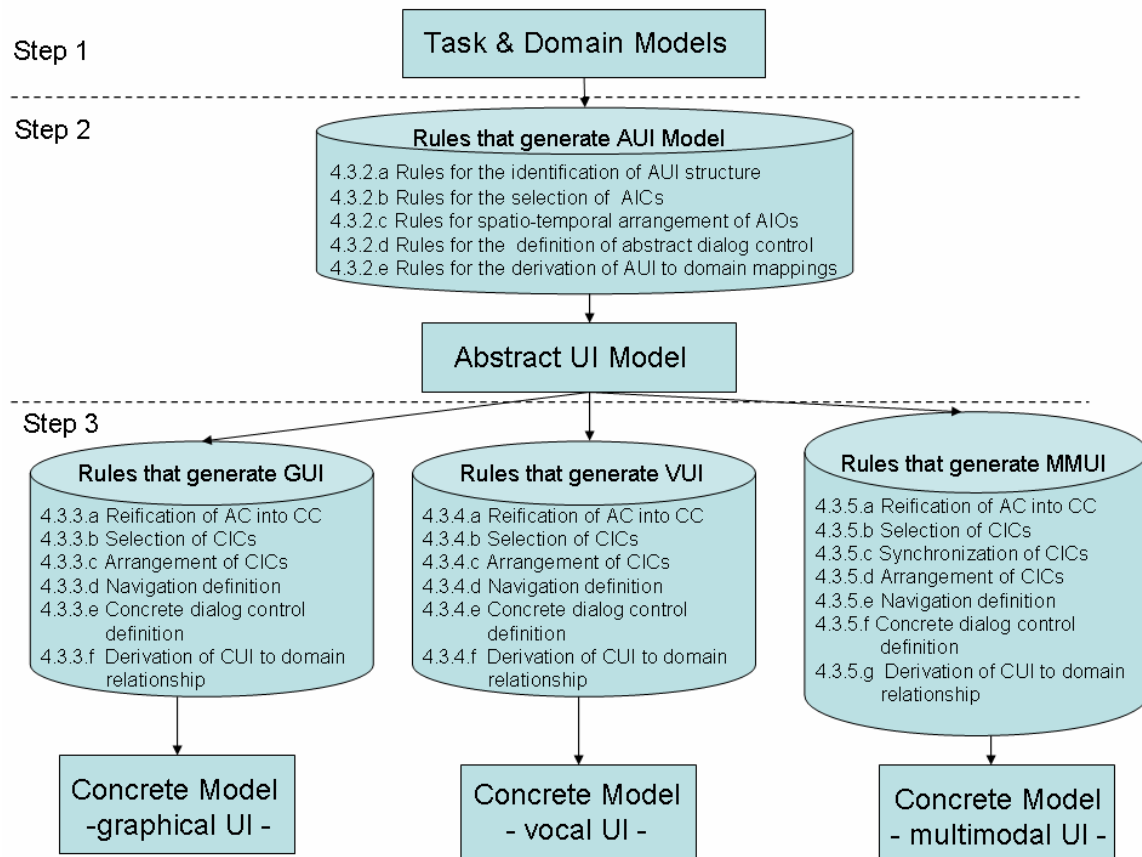


Figure 4-40 Sub-steps of the transformational approach

4.4.1 Step 1: The Task and Domain Models

The initial development step consists of specifying the Task and Domain Models which requires first the identification of the interactive tasks along with their associated attributes and second the specification of the relationships between the tasks. The Domain Model consists of identifying the classes and their corresponding attributes and methods manipulated by the user while interacting with the system. Domain relationships between classes are further established by specifying their role names and cardinalities. Once the Task and Domain Models are specified, the mappings between them can be identified. Each task from the Task Model will be mapped into a corresponding element from the Domain Model.

4.4.2 Step 2: From Task and Domain Models to Abstract User Interface Model

The second transformation step involves a transformation system containing rules applied in order to realize the transition from the Task and Domain Models to Abstract UI Model. It consists of the five development sub-steps illustrated in Figure 4-40 applied in top-down logical order.

4.4.2.a Rules for the identification of AUI structure

This sub-step consists of defining groups of AIOs that correspond to groups of tasks tightly coupled together (e.g., the children of the same task can be considered as a group of tightly coupled tasks). For this purpose, the following design options are considered:

Sub-task presentation. Enables the identification of the ACs depending on the following design decision:

- *Separated* (Figure 4-41): Rule 1 generates an AC for each sub-task of the root task (AC11, AC12 and AC13).
- *Combined* (Figure 4-42): Rule 1 creates the AC (AC1) in which the root task is executed and Rule 3 generates an AC for each sub-task of the root task (AC11, AC12 and AC13).

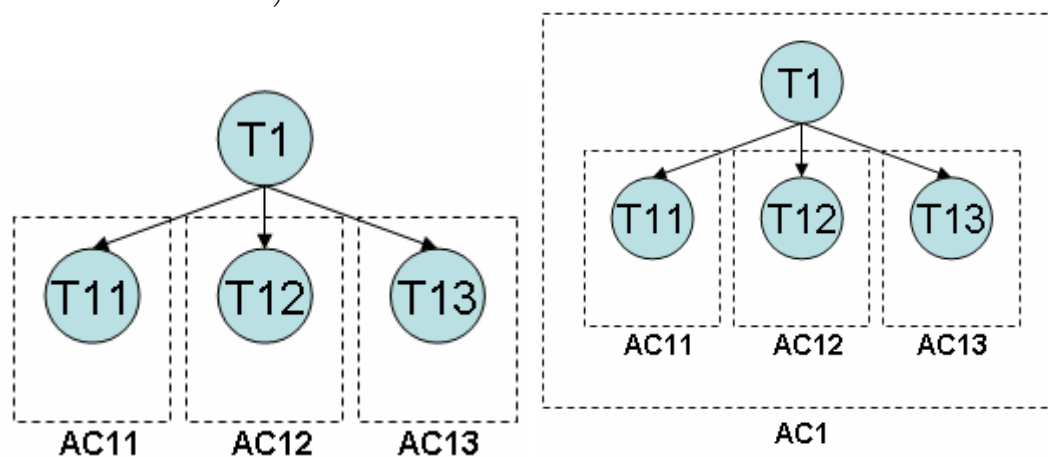


Figure 4-41 Separated sub-task presentation

Figure 4-42 Combined sub-task presentation

Navigation type. Considers the containment and the cardinality of the AICs ensuring the navigation according to the following values:

- *Local* (Figure 4-43): Rules 27, 28 and 29 generate one/two AICs that ensure the navigation depending on the position of the AC in the abstract tree structure (AIC111, AIC121, AIC122, AIC131). Each AIC is embedded into the corresponding AC (i.e., AC1, AC2 and AC3).
- *Global* (Figure 4-44): Rule 34 generates two global placed AICs (i.e., AIC11, AIC12) embedded into the top-most AC (i.e., AC1).

The possible values for the cardinality are:

- *Simple* (Figure 4-43): Rule 37 creates two AICs.
- *Multiple*: Rules 38 and 39 generate AICs contained locally and globally in their corresponding ACs in order to ensure redundant navigation capabilities.

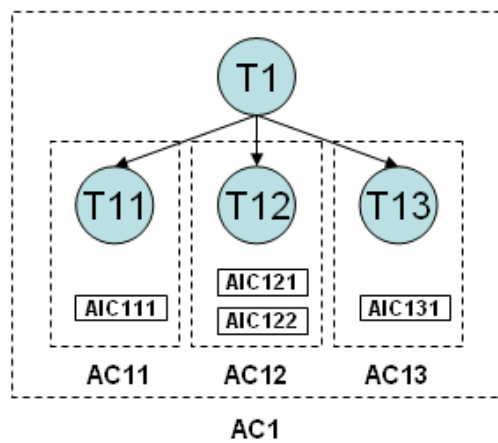


Figure 4-43 Local placement for navigation

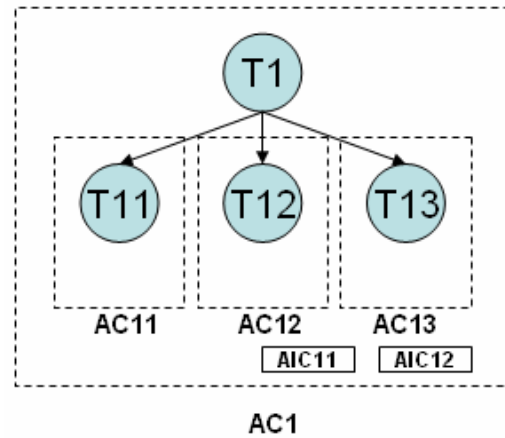


Figure 4-44 Global placement for navigation

Control type. Considers the containment and the cardinality of the AICs ensuring the control of data. The possible values for the containment are:

- *Local* (Figure 4-45): Rule 40 generates two AICs (i.e., AIC111, AIC112, AIC121, AIC122, AIC131 and AIC132) that ensure the control of data for each sub-task executed into a corresponding AC (i.e., AC1, AC2 and AC3).
- *Global* (Figure 4-46): Rule 43 generates two global placed AICs (i.e., AIC11, AIC12) embedded into the top-most AC (i.e., AC1).

The possible values for the cardinality are:

- *Simple* (Figure 4-45 and Figure 4-46): Rule 46 creates two AICs that will be concretized in two logically connected buttons (i.e., OK, CANCEL).
- *Multiple*: Rules 47 and 48 create AICs placed locally and globally in their corresponding ACs in order to ensure redundant control of data.

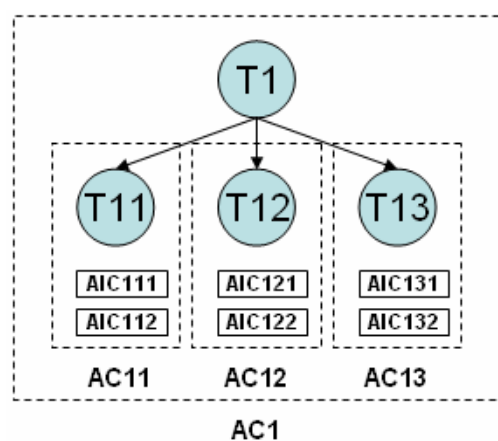


Figure 4-45 Local placement for control

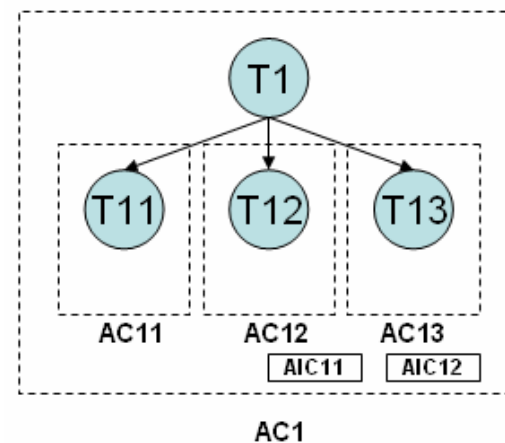


Figure 4-46 Global placement for control

4.4.2.b Rules for the selection of AICs

The goal of this sub-step is to produce the specification of the AICs. As AICs assume basic interaction functions through facets, our objective is limited to their proper selection. In order to achieve this goal, we consider the information contained in the

4. A Transformational Method for Producing Multimodal User Interfaces

Task and Domain Models, in particular the *taskType* and *taskItem* attributes of a task along with the *manipulates* relationship that offers more information about the domain concepts manipulated by the task.

Table 4-2 provides the mappings between the types and items manipulated by a task and their correspondent in the Abstract Model (Figure 3-5). Due to the existence of a great number of combinations of *taskTypes* and *taskItems*, the table is restricted only to a small subset used in the current thesis.

Task taskType + taskItem	AIC facet facet type + (actionType + actionItem)
Start + operation	Control + (start + operation)
Start + operation	Navigation + (start + operation)
Select + element	Input + (select + element)
Create + element	Input + (create + element)
Convey + element	Output + (convey + element)

Table 4-2 Mappings between tasks types and AIC facets types

The selection of several AICs is supported by the following design options which identify the type of facets to generate:

Navigation type. Rule 30 creates a *navigation* facets of type *start operation* if the task corresponding to its AIC is of type *start operation* and *manipulates* a method from the Domain Model.

Control type. Rule 41 creates a *control* facet of type *start operation* if the task corresponding to its AIC is of type *start operation* and *manipulates* a method from the Domain Model.

Navigation and control type. Two design values could be considered:

- *Separated:* Rule 49 endows AICs embedded in the same AC with *navigational* and *control* facets, respectively.
- *Combined:* Rule 50 generates *navigational* and *control* facets for the same AIC.

However the selection of AICs is not always supported by a design option value. This is the case of Rule 81 which generates an *output* facet of type *convey element* for each task that supposes a *convey* action from the part of the system. The task is manipulating an attribute from the Domain Model.

4.4.2.c Rules for spatio-temporal arrangement of AIOs

This sub-step ensures the arrangements of objects that populate the AUI by specifying the layout constraints between the AIOs. These constraints are derived from the Task Model structure. The order in which the tasks are specified allow designers to determine the order in which the AIOs are conveyed. For this purpose, the *abstractAdjacency* relationship is employed.

For each couple of sister tasks executed into AIOs, we define *abstractAdjacency* relationships between them. As AIOs can be of two types (i.e., ACs or AICs) there are four possible combination to consider. For each of them a specific rule is applied. Moreover, in order to perform a complete arrangement, a rule should be defined for each type of temporal relationships between the tasks. For instance, Rules 87-90 illustrate the

4. A Transformational Method for Producing Multimodal User Interfaces

generation of *abstractAdjacency* relationships between couples of AIOs mapped into sibling tasks connected by sequential (“>>”) temporal relationship.

4.4.2.d Rules for the definition of abstract dialog control

This sub-step is transposing the temporal relationships defined between tasks into abstract relationships between AIOs. The dialog control [Limb04b] expresses the locus of control (i.e., availability) for initiating the dialog in a UI. It refers to the control of certain states of the UI in order to enforce temporal constraints imposed between elements of the interface.

In order to ensure the abstract dialog control we employ the *auDialogControl* relationship. For each couple of sibling tasks executed into AIOs, we define an *abstractDialogControl* relationship between them that have the same semantics as the temporal relationship defined between the tasks. As AIOs can be of two types (i.e., ACs or AICs), there are four possible combinations to consider. For instance, Rules 91-94 illustrate the generation of *auDialogControl* relationships between couples of AIOs mapped into sibling tasks with temporal dependencies.

4.4.2.e Rules for the derivation of AUI to domain mappings

This sub-step consists of refining the *manipulates* relationship defined between elements of the Task Model and elements of the Domain Model into relationships between AICs from Abstract Model and elements of the Domain Model. The two refined relationship considered in the current dissertation are *updates* and *triggers*.

For instance, Rule 95 is employed in order to refine *manipulates* relationship between a task and an attribute from the Domain Model into *updates* relationship between the AIC in which the task is executed and the above mentioned attribute. Rule 96 is employed in order to refine *manipulates* relationship between a task and a method from the Domain Model into *triggers* relationship between the AIC in which the task is executed and the above mentioned method.

4.4.3 Step 3: From Abstract User Interface Model to Concrete User Interface Model

The third transformation step consists of a set of development sub-steps that contains transformation rules applied in order to achieve the transition from the Abstract UI Model to the Concrete UI Model. Depending on the considered interaction modality different rules have to be applied.

4.4.3.a Selection of modality

The Concrete UI Model aims to define a UI that is dependent of the interaction modality but independent of any software platform. It is now that the designer selects the available modalities employed in order to enable the interaction between the system and the user. Therefore, three cases have been identified (Figure 4-39):

- *Case 1:* From AUI Model to Graphical CUI Model: only the graphical modality is available in input and in output.
- *Case 2:* From AUI Model to Vocal CUI Model: only the vocal modality is available in input and in output.

4. A Transformational Method for Producing Multimodal User Interfaces

- *Case 3:* From AUI Model to Multimodal CUI Model: graphical and/or vocal modalities are available in input and in output.

4.4.3.b Design option for the selected modality

For each type of Concrete UI a specific set of transformation sub-steps are considered. Moreover, for each sub-step we identify hereafter the design options to consider in order to decide between the different design features of the UIs.

4.4.3.b.1 Case 1: From AUI Model to Graphical CUI Model

The current case derives Graphical Concrete UIs from Abstract UI specifications by applying a set of transformation rules structured in six development sub-steps (Figure 4-40). The transformation supporting the current case consider only the abstract and graphical concrete concepts illustrated in black and red, respectively.

4.4.3.b.1.1 Reification of AC into CC

This sub-step is dedicated to the reification of ACs into GCs. In Section 4.4.2.a the identification of ACs considered the **Sub-task presentation** design option. As the Concrete Model is modality-dependent, their values are concretized in the current section in graphical objects according to Figure 4-3. Hereafter, we present two design option values and identify the corresponding rules in the Transformation Catalog:

- *Separated:* Rule 2 generates for each top-most AC a GC of type *window*.
- *Combined all at once in grouped list:* first, Rule 17 reifies the top most AC into a *window* containing a *box* and further Rule 18 generates a *groupBox* for each AC embedded into the top-most AC.

4.4.3.b.1.2 Selection of CICs

This sub-step supposes the identification of graphical concrete elements that are suitable to support the functionalities of AICs ensured by their facets identified in Section 4.4.2.b. provides mappings between AICs defined by their facets and GICs that reify them. Due to the great number of combinations of task types and items, Table 4-3 is restricted only to a small subset used in the current dissertation. The left column identifies the combinations of *actionType* and *actionItem* attributes of AIC facets, the middle column shows the corresponding GIC, whereas the right column specifies the rule(s) applied to generate these GICs.

AIC facet facet type + (actionType + actionItem)	GIC type	Transformation rule
Control + (start + operation)	button	Rules 42, 45
Navigation + (start + operation)	button	Rules 31, 32, 33, 36
Input + (select + element)	radioButton	Rules 99, 100
Input + (select + element)	checkBox	Rules 101, 102
Input + (select + element)	comboBox	Rules 97, 98
Input + (select + element)	listBox	Rules 103, 104
Input + (create + element)	inputText	Rules 105
Output + (convey + element)	outputText	Rules 106

Table 4-3 Mappings between facet types and GIC types

4. A Transformational Method for Producing Multimodal User Interfaces

For this purpose, the following design options are considered: **Prompting, Input, Immediate feedback, Guidance, Sub-task guidance, Answer cardinality, Confirmation answer and Answer order**. An exemplification of their values is presented based on a possible design decision for a text input where the user provides graphically his/her name. Table 4-4 identifies the rules applied in order to generate the corresponding GICs.

Design option	Value	GIC	Transformation rule
Prompting	Graphical (A)	outputText	Rule 66
Input	Graphical (A)	inputText	Rule 60
Immediate feedback	Graphical (A)	inputText	Rule 69
Guidance for input	Textual (A)	outputText	Rule 71
Guidance for feedback	-	-	-
Sub-task guidance	Unguided	-	-
Answer cardinality	Simple	inputText	Rule 60
Confirmation answer	Without confirmation	-	-
Answer order	-	-	-

Table 4-4 Design option values for textInput widget with graphical assignment for input

4.4.3.b.1.3 Arrangement of CICs

This sub-step is applied in order to provide the concrete layout information of the UI. It consists of a transposition of the *abstractAdjacency* relationship defined between each couple of AIOs (Section 4.4.2.c) into a *graphicalAdjacency* relationship between GIOs reifying them. As AIOs can be of two types (i.e., ACs or AICs), four rules describing the four possible combinations are considered (Rules 121-124).

4.4.3.b.1.4 Navigation definition

This sub-step aims to specify the navigation structure among the different GCs populating a UI. In Section 4.4.2.a the generation of AIC that ensures the navigation between containers was based on the **Navigation type** design option. In Section 4.3.3.b.1.2 the AICs were reified in their corresponding GIC (i.e., “PREV” and “NEXT” buttons).

The current sub-step considers the **Sub-task navigation** design option. It enables to define the navigation type between GCs by endowing the GICs that ensure the navigation with graphical transition features. There are two possible values of this option:

- *Sequential*: for exemplification we consider three sub-tasks executed in *separated windows* (Figure 4-47). The navigation between the windows is ensured by the *PREV* and *NEXT buttons*. Once the user fulfilled the requested information in the window corresponding to sub-task 1, only sub-task 2 can be activated (navigation *a*). From the window associated to sub-task 2, there are two paths: returning to sub-task 1 window (navigation *b*) or continuing with sub-task 3 (navigation *c*).

4. A Transformational Method for Producing Multimodal User Interfaces

From sub-task 3 window the user can activate only sub-task 2 window (navigation *d*). Rule 23 offers the support for this value.

- *Asynchronous* (Figure 4-48): the navigation is ensured by *buttons* that specify the name of the sub-task they activate. Once the user fills in the requested information in the window corresponding to sub-task 1, either sub-task 2 or sub-task 3 can be activated (navigation *a* and *b*, respectively). From the window associated to sub-task 2 the user can navigate back to sub-task 1 (navigation *c*) or continue to fill in the information requested for sub-task 3 (navigation *d*). In the window associated to sub-task 3, there are two navigational paths: returning to sub-task 2 (navigation *e*) or navigating to sub-task 1 (navigation *f*). Rules 24, 25 and 26 ensure the support for this value.

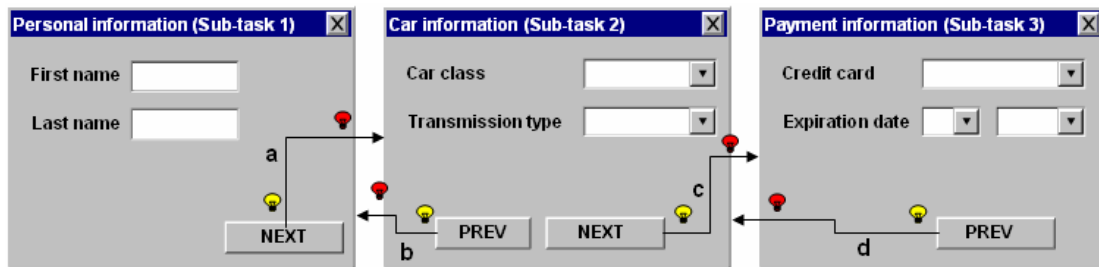


Figure 4-47 Sequential navigation between sub-tasks presented in separated windows

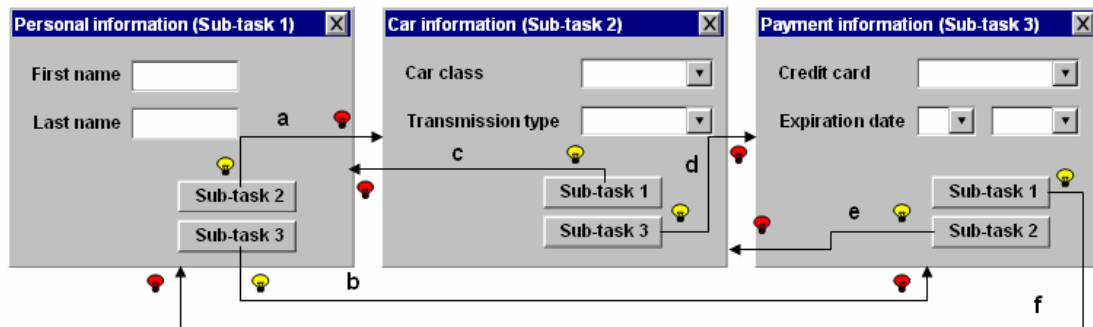


Figure 4-48 Asynchronous navigation between sub-tasks presented in separated windows

4.4.3.b.1.5 Concrete dialog control definition

This sub-step realizes a transposition of *aniDialogControl* relationships defined between each couple of AIOs into *cuiDialogControl* relationships between graphicalCIOs reifying them. As AIOs are of two types (i.e., ACs and AICs), four rules describing the four possible combinations are considered (Rule 125-128).

4.4.3.b.1.6 Derivation of CUI to domain relationships

This step aims at transposing the relationships defined in Section 4.4.2.e to the concrete level. Thus, relationships between GICs and domain objects are defined thanks to Rules 129 and 130.

4.4.3.b.2 Case 2: From AUI Model to Vocal CUI Model

The current case aims at deriving Vocal Concrete UIs from Abstract UI specifications by applying a set of transformation rules structured in six development sub-steps (Figure

4. A Transformational Method for Producing Multimodal User Interfaces

4-40). The transformation supporting the current case consider only the abstract and vocal concrete concepts illustrated in black and blue, respectively.

4.4.3.b.2.1 Reification of AC into CC

This sub-step is dedicated to the reification of AC into VC. By analogy with Section 4.3.3.b.1.1, the current sub-step considers the *sub-task presentation* design option. As the Concrete Model is modality-dependent their values are concretized in vocal objects according to Figure 4-3. Hereafter, we present two design option values and we identify the corresponding rules in the transformation catalog:

- *Separated*: Rule 2 generates for each top-most AC a VC of type *vocalGroup*.
- *Combined all at once in grouped list*: first, Rule 17 reifies the top most AC into a *vocalGroup* containing a *vocalForm* with a *vocalInput* and then Rule 18 generates a *vocalGroup* and two *vocalPrompts* for each AC embedded into the top most AC.

4.4.3.b.2.2 Selection of VICs

By analogy with Section 4.4.2.b, we provide in Table 4-5 the mappings between AICs defined by their facets and the VICs that reify them. The left column identifies the combinations of *actionType* and *actionItem* attributes of AIC facets, the middle column shows the corresponding VIC type, whereas the right column specifies the transformation rule to apply in order to generate the VICs.

AIC facet facet type + (actionType + actionItem)	VIC type	Transformation rule
Control + (start + operation)	submit	Rules 42, 45
Navigation + (start + operation)	vocalNavigation	Rules 31, 32, 33, 36
Input + (select + element)	vocalInput + grammar + part + items	Rules 97, 98 or Rules 99, 100 or Rules 101, 102 or Rules 103, 104
Input + (create + element)	vocalPrompt + vocalInput + record	Rule 105
Output + (convey + element)	vocalPrompt	Rule 106

Table 4-5 Mappings between facet types and VIC types

By analogy with the correspondent sub-step in Case 1, the designer takes into consideration the following design options: **Prompting, Input, Immediate feedback, Guidance, Sub-task guidance, Answer cardinality, Confirmation answer and Answer order.**

We exemplify the design options considered in the current sub-step with a possible design decision for a vocal input that enables users to utter their names. Table 4-6 identifies the rules applied in order to generate the corresponding VICs.

Design option	Value	VIC	Transformation rule
Prompting	Vocal (A)	vocalPrompt	Rule 65
Input	Vocal (A)	vocalInput	Rule 59
Immediate	Vocal (A)	vocalFeedback	Rule 68

4. A Transformational Method for Producing Multimodal User Interfaces

feedback			
Guidance for input	Speech (A)	vocalPrompt	Rule 74
Guidance for feedback	-	-	-
Sub-task guidance	Unguided	-	-
Answer cardinality	Simple	vocalInput	Rule 59
Confirmation answer	Without confirmation	-	-
Answer order	-	-	-

Table 4-6 Design option values for vocal assigned input

Note: for the rest of the sub-steps: Arrangement of CICs, Navigation definition, Concrete dialog control definition, Derivation of CUI to domain relationships, we are employing the abstract and the vocal concepts of the rules presented in Section 4.4.3.b.1.

4.4.3.b.3 Case 3: From AUI Model to Multimodal CUI Model

The current case derives MM Concrete UIs from Abstract UI specification by applying a set of transformational rules structured in seven development sub-steps (Figure 4-40). The transformation supporting the current case consider the abstract and both graphical and vocal concepts illustrated in black, red and blue, respectively.

4.4.3.b.3.1 Reification of AC into CC

As described in the homologous sub-steps of Cases 1 and 2, the rules that ensure the current sub-step consider the different possible final representations of the sub-task presentation design option.

Sub-task presentation. Hereafter, we present two design option values and we identify the corresponding rules in the transformation catalog:

- *Separated:* Rule 2 generates for each top-most AC a *window* and a *vocalGroup*.
- *Combined all at once in grouped list:* first, Rule 17 reifies the top most AC into a *window* and a *vocalGroup* containing a *vocalForm* with a *vocalInput* and further Rule 18 generates a *groupBox*, a *vocalGroup* and two *vocalPrompts* for each AC embedded into the top most AC.

4.4.3.b.3.2 Selection of CICs

In order to identify the MM CICs that are the most suitable to support the functionalities of the AICs ensured by their facets, Table 4-7 provides a series of mappings used in this thesis.

AIC facet facet type + (actionType + actionItem)	GIC and VIC types	Transformation rules
Control + (start + operation)	button + submit	Rules 42, 45
Navigation + (start + operation)	button + vocalNavigation	Rules 31, 32, 33, 36
Input +	radioButton + vocalInput +	Rules 99, 100

4. A Transformational Method for Producing Multimodal User Interfaces

(select + element)	grammar + part + items	
Input + (select + element)	checkBox + vocalInput + grammar + part + items	Rules 101, 102
Input + (select + element)	comboBox + vocalInput + grammar + part + items	Rules 97, 98
Input + (select + element)	listBox + vocalInput + grammar + part + items	Rules 103, 104
Input + (create + element)	inputText + vocalPrompt + vocalInput + record	Rules 105
Output + (convey + element)	outputText + vocalPrompt	Rules 106

Table 4-7 Mappings between facet types and GIC and VIC types

By analogy with the correspondent sub-step in Case 1, the designer takes into consideration the following design options: **Prompting, Input, Immediate feedback, Guidance, Sub-task guidance, Answer cardinality, Confirmation answer and Answer order**. An exemplification of their values is presented based on a possible design decision (Figure 4-18) for a MM text input where the user has to provide his/her name. Table 4-8 identifies the rules applied in order to generate the corresponding CICs.

Design option	Value	CIC	Transformation rule
Prompting	Multimodal (R)	outputText + vocalPrompt	Rule 67
Input	Multimodal (E)	inputText + vocalInput	Rule 61
Immediate feedback	Multimodal (R)	inputText + vocalFeedback	Rule 70
Guidance for input	Iconic (assignment)	imageComponents (keyboard icon + microphone icon)	Rule 72
Guidance for feedback	Iconic (assignment)	imageComponent (speakerIcon)	Rule 77
Sub-task guidance	Unguided	-	-
Answer cardinality	Simple	vocalInput	Rule 59
Confirmation answer	Without confirmation	-	-
Answer order	-	-	-

Table 4-8 Design option values for multimodal textInput widget (graphical and vocal equivalence for input)

4.4.3.b.3.3 Synchronization of CICs

Unlike the previous two cases the current one adds introduces a new sub-step (Requirement 10. Method extendibility) aiming at ensuring the coordination of vocal and graphical CIOs by generating a synchronization relationship between them (Section 3.4.4). Hereafter, we identify the rules supporting this sub-step for two examples:

- If the designer wants to enable users to interact with a combobox widget by employing the vocal modality, then one must ensure the synchronization between the *vocalInput* that will gather the input from the user and the *comboBox*: Rule 116

4. A Transformational Method for Producing Multimodal User Interfaces

defines the synchronization between the *currentValue* x of the *vocalInput* and the *currentValue* z of the *comboBox*.

- The second example corresponds to the designer's decision of allowing users to interact vocally with a text field. Thus, synchronization between the *vocalInput* that gathers user's input and the *inputText* is ensured by Rule 120 that synchronizes the *currentValue* x of the *vocalInput* and the *currentValue* z of the *inputText*.

Note: for the rest of the sub-steps: 4.4.3.b.2.3 Arrangement of CICs, 4.4.3.b.2.4 Navigation definition, 4.4.3.b.2.5 Concrete dialog control definition, 4.4.3.b.2.6 Derivation of CUI to domain relationships, we are employing the abstract, the vocal and the graphical concepts of the rules presented in Section 4.4.3.b.1.

4.4.4 Step 4: From Concrete User Interface Model to Final User Interface

This step generates the source code of the FUI from each type of CUI considered in the previous step (i.e., graphical, vocal and MM). Thus, for GUIs we generate XHTML code, VoiceXML code is considered for VUIs, while MM UIs will be supported by XHTML+Voice language. Further, we interpret the generated code within a corresponding browser:

- Graphical FUI: any ordinary web browser (e.g., Internet Explorer, Mozilla).
- Vocal FUIs: interpreted with IBM VoiceXML browser.
- Multimodal FUIs: interpreted within Opera browser. With respect to the CARE properties we consider only *Assignment*, *Equivalence* and *Redundancy in output*. *Redundancy in input* and *Complementarity in input/output* are not covered as fusion and fission are not currently supported by the X+V language and consequently we do not have any control over these aspects. Moreover, they are out of the scope of this thesis (Section 1.4.3).

4.5 Conclusion

This chapter introduced the design space and expanded the selected transformational approach with the concept of colored transformation rules gathered in a transformation catalog. Based on the mappings between the design option values and the transformation rules we were able to identify and exemplify the design options supporting each development sub-step. Thanks to this identification in Chapter 6 we will prove the feasibility of developing different MM UIs for which a large variety design decisions were considered.

5 Tool Support

5.1 Introduction

One of the main advantages of our design space is given by the fact that each design option composing it is independent of any existent method and tool, thus being useful for any developer of MM UIs. In these circumstances, an explicit support of the introduced design space offered by a tool that implements the proposed method would be a real help for designers (Requirement 11. Machine processability of involved models). Therefore, we consider MultimodalXML, an assembly of five software modules for computer-aided design of MM UIs [Stan06] .

The tool is reducing the designer's set of concerns by limiting the amount of design decisions to those composing our design space, thus providing a more manageable and tractable solution [Hoov91]. Based on the transformational approach general development scenario we identified the five modules of MultimodalXML tool over the steps in which they are employed (Figure 5-1). The tool ensures interoperability as the result produced by one module can be reused in another module (Requirement 16. Support for tool interoperability).

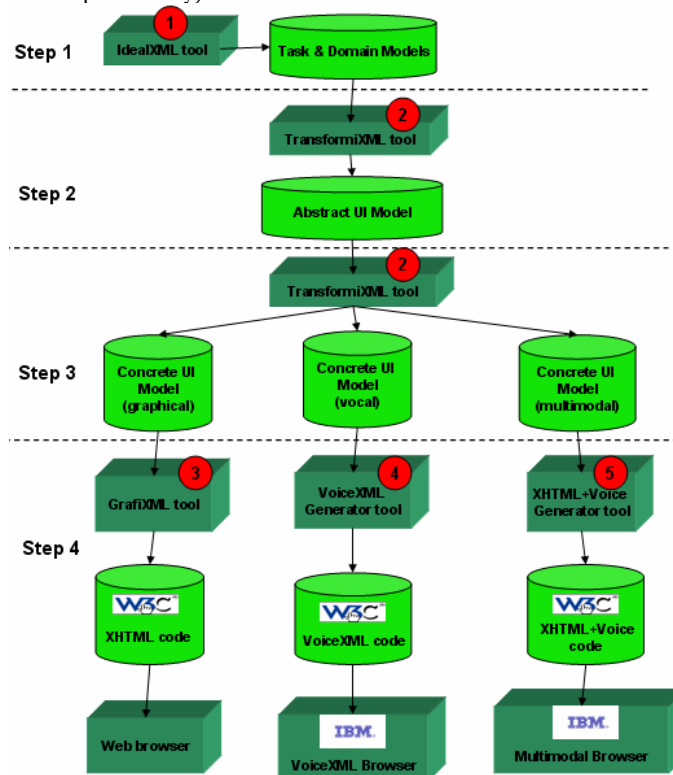


Figure 5-1 General development scenario – identification of MultimodalXML modules

5.2 MultimodalXML modules

This section presents each module according to the following structured schema: (1) a detailed description of the features is provided, (2) the implementation characteristics (e.g., programming language, libraries employed, author) are specified, (3) our contribution module is emphasized.

5.2.1 IdealXML

Description. IdealXML [Mont05] is a tool that is involved in the first step of the transformational approach and allows designers to describe graphically the Task and Domain Models and the mappings between them. Moreover, the tool enables to graphically specify the Abstract UI Model, but due to the fact that the main objective of UsiXML is to provide a machine processable language and then a human readable specification, in this dissertation we generate the Abstract Model by employing the transformational approach. The tool is able to automatically generate the UsiXML specification of the corresponding models.

The Task Model (Figure 5-2) takes the form of a CTT notation [Pate97]. The Domain Model (Figure 5-3) has the appearance of a class diagram, while the Mapping Model (Figure 5-4) is specified by associating graphically elements of the Task Model with elements of the Domain Model.

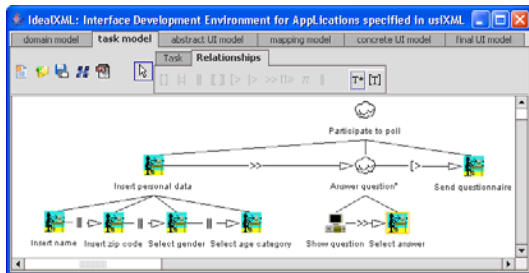


Figure 5-2 Task Model editor

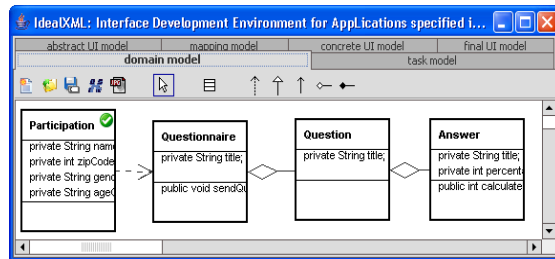


Figure 5-3 Domain Model editor

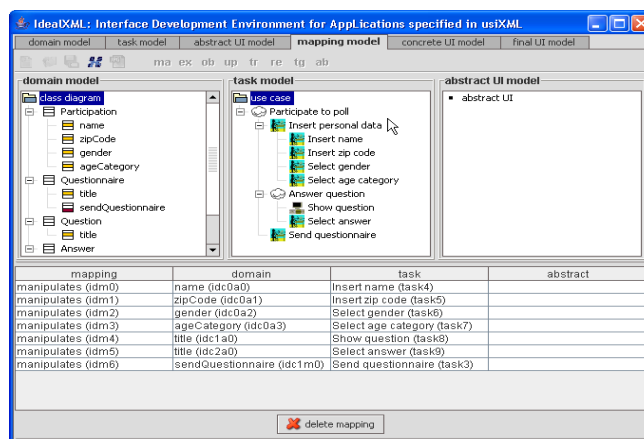


Figure 5-4 Mapping Model editor

Implementation. The tool is implemented in Java language by Francisco Montero.

Contribution. Our contribution to this tool concerns more the conceptual aspects. It consists of the introduction of an expanded Task Model (Section 3.3.1.b) with features

5. Tool support

that respond to the requirements of MM UIs. However, these contributions have not been implemented in the tool yet.

5.2.2 TransformiXML

Description. The transformation approach is sustained in steps 2 and 3 by TransformiXML, the core module of the MultimodalXML software that enables the definition and the application of transformation rules based on design options. The basic flow of tasks with TransformiXML GUI (Figure 5-5) is the following: after choosing an input file containing models to transform, the user selects a development path by choosing a starting point (i.e., the initial model) and the destination point (i.e., the model to reach). All the steps and sub-steps of the chosen path can be visualized in the *development path explorer* frame. By clicking on a sub-step, a set of transformation systems realizing the chosen sub-step are displayed in the transformation system explorer. Each transformation system contains the corresponding rules described in the transformation rule catalog that can be visualized in the *transformation rule explorer* frame. Depending on the considered design option, the designer will select the correspondent transformation(s). The designer is also able to edit the rules either in GrafiXML editor or in AGG tool [Ehri99]. The result of the transformation is then explicitly saved in a UsiXML file.

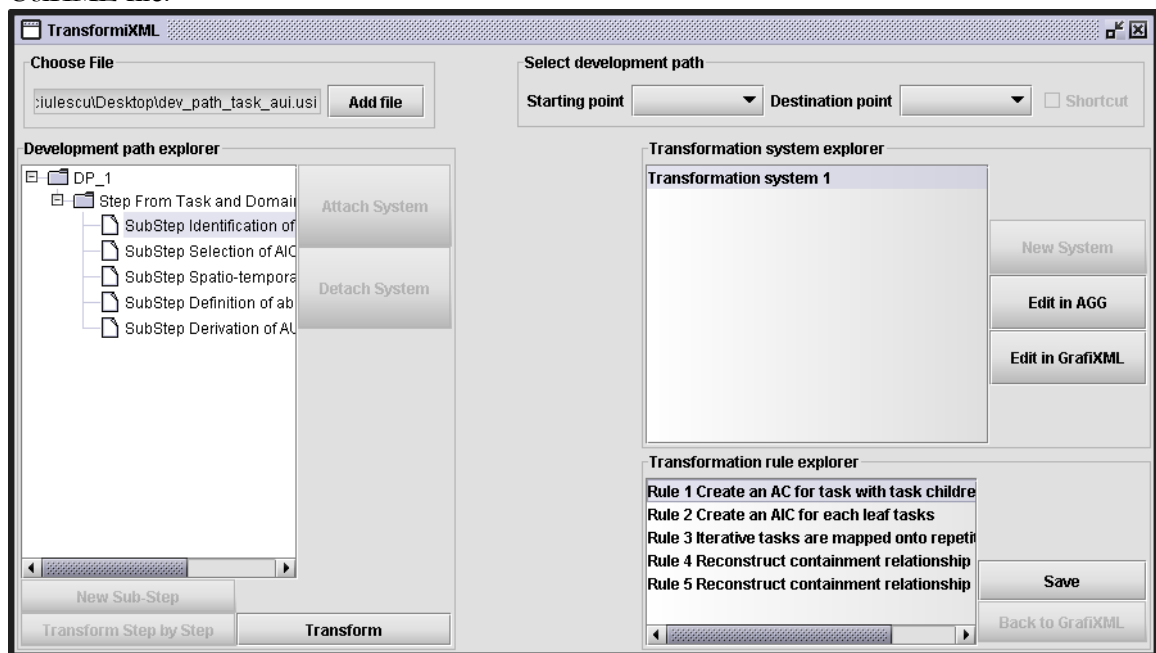


Figure 5-5 TransformiXML – graphical user interface

Implementation. TransformiXML is developed jointly by Quentin Limbourg, Victor Lopez-Jaquero and Benjamin Michotte in Java programming language by employing the AGG API that was selected due to our prior experience with the AGG tool. AGG is an open-source development environment for attributed graph transformation systems supporting an algebraic approach to graph transformation [Ehri99]. The scenario of using AGG API to perform model-to-model transformations consists of the following phases (Figure 5-6): the initial specification of a model along with a set of rules both expressed

5. Tool support

in UsiXML are processed by the TransformiXML API. A parsing operation is applied over the UsiXML elements (models and rules) which are transformed into AGG objects. The set of rules are applied sequentially to the models in order to obtain the resultant AGG objects. Further, the objects are parsed and transformed into UsiXML resultant specification.

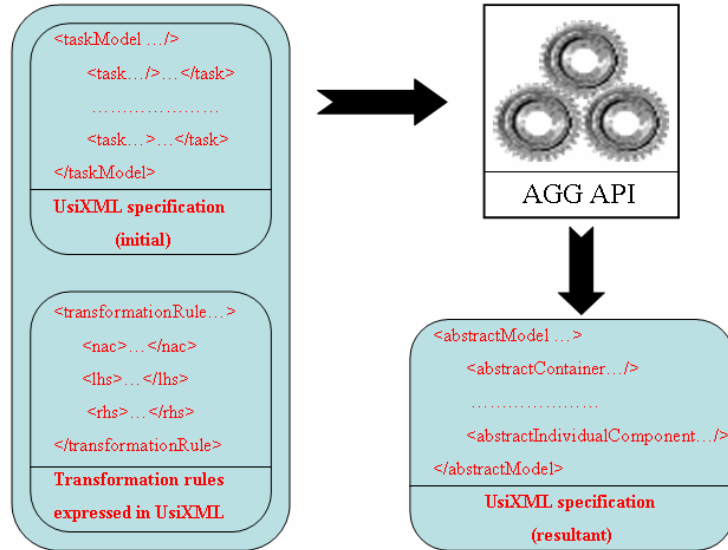


Figure 5-6 Model-to-model transformation based on AGG API

TransformiXML tool has been tested successfully on a series of examples, but for the moment it does not support the automatic application of transformation rules for all the steps and sub-steps involved in the transformational method. However, the feasibility of the approach was proved to be successful in model-to-model transformation generated manually with AGG tool. Figure 5-7 provides an example of a transformation rule applied manually over the initial *Task Model* (Figure 5-8) in order to generate the resultant AUI Model (Figure 5-9). The rule is creating AC in which each sub-task of the top-most task in a *Task Model* will be executed.

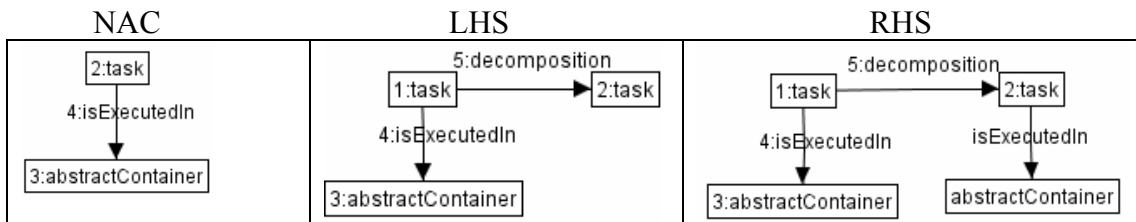


Figure 5-7 Generate abstract containers for each sub-task of the top-most task

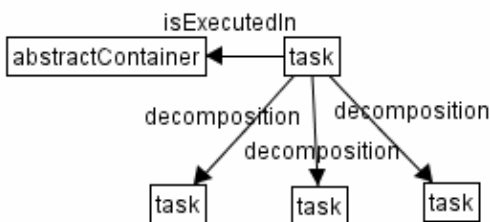


Figure 5-8 Initial Model

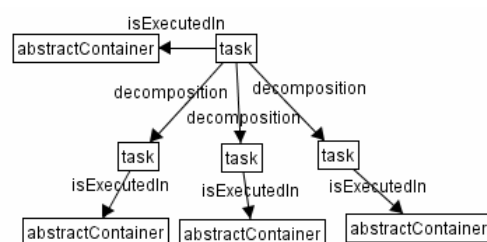


Figure 5-9 Resultant model

5. Tool support

Contribution. The contributions brought to this module are two fold:

- **Conceptual contributions:** previous to our work the tool was employed in editing and applying transformation for the generation of GUIs [Limb04b]. This work enriches the existing transformational approach method in order to support generation of vocal and MM UIs. This is made possible thanks to the introduction of new sub-steps (i.e., *Synchronization between CICs* in Section 4.4.3.b.3.3) that involve new transformation rules defined over an expanded vocal ontology (Section 3.4.2)
- **Implementation contribution:** we have implemented import and export functionalities in AGG as part of an incipient project involving TransformiXML tool [Stan04]. The import functionality allows to represent under the form of a graph the XML specification corresponding to any level of UsiXML language, whereas the export functionality enables designers to recover the resultant graph under the form of XML specification of UsiXML. Moreover, we have ensured the testing phase during the continuous development of the tool which enabled designers to improve its implementation thanks to the identified bugs.

5.2.3 GrafiXML

Description. GrafiXML is a tool that is involved in Step 4 of the transformational approach. It allows designers to import the graphical CUI specification obtained in the previous step and to export it into XHTML code (Figure 5-10). GrafiXML can also be used to enable the development of CUI Models by designers. For this purpose a specific editor has been developed where the designers can draw in direct manipulation any GUI by placing graphicalCUIOs and editing their properties in a property sheet. The correspondent UsiXML specification can be visualized and modified at any moment, while the changes are being updated immediately into the graphical representation.

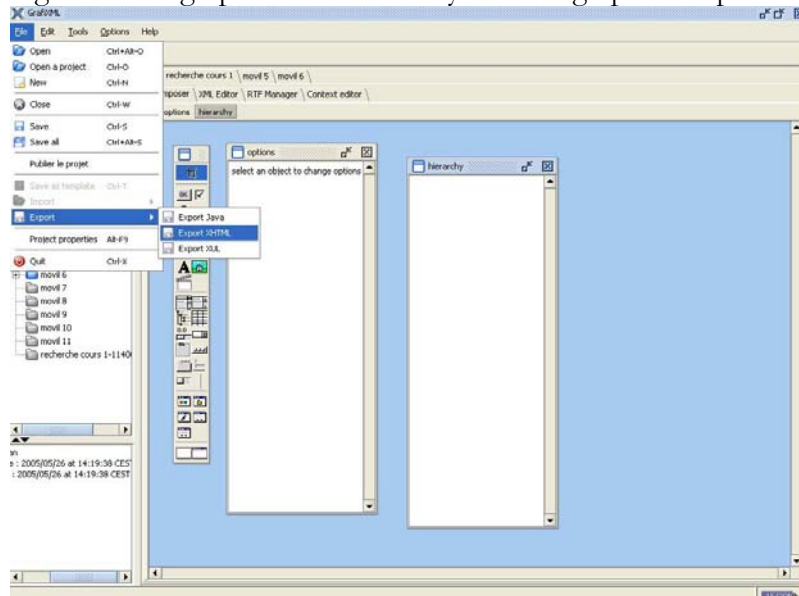


Figure 5-10 GrafiXML – export function

Implementation. GrafiXML is developed by Benjamin Michotte in Java language and requires the following libraries: Java Help jars, mysql.jar, Java Web Start jars, jdom.jar,

5. Tool support

Castor-x.y.jar, oro.jar, commons-logging.jar, xerces.jar, regexp.jar, Java Media Framework.

Contribution. We did not bring any contribution to the development of this tool as it addresses the development of graphical interaction only, whereas our work consisted more in expanding the the vocal and MM aspects. However, the tool could be extended in order to support editing vocal and MM CUIs by simply enabling to graphically manipulate the vocal concepts introduced by our ontology and by defining relationships with graphical objects.

5.2.4 VoiceXML Generator

Description. VoiceXML Generator tool is a module involved in step 4 of the transformational approach. It generates VoiceXML code by applying XSL Transformations [Clar99] over the vocal specification of the CUI Model. These transformations are capable of creating, inserting, updating, deleting or replacing fragments of any XML-compliant languages like HTML, XUL, XIML, UIML and of course UsiXML. Using XSL transformations, rich behavior can be produced in order to generate the final UI.

Implementation and contribution. So far, there is no implementation for this tool. The transformations were applied manually by mapping the vocal CIOs into the corresponding VoiceXML elements. Table 5-1 provides several mapping examples.

UsiXML vocal CIO	VoiceXML element
vocalForm	form
vocalPrompt	block
vocalInput	field
vocalMenu	menu
vocalMenuItem	choice
grammar	grammar
part	rule
vocalNavigation	goto

Table 5-1 Mappings between the vocal CIOs and VoiceXML elements

5.2.5 XHTML+Voice Generator

Description. XHTML+Voice Generator tool is a module involved in step 4 of the transformational approach. It generates XHTML+Voice code by applying XSL Transformations over the MM specification of the CUI Model.

Implementation and contribution. So far, there is no implementation for this tool. The transformations were applied manually by mapping the graphical and vocal CIOs into the XHTML and VoiceXML elements, respectively. Table 5-1 and Table 5-2 provide several mapping examples.

UsiXML graphical CIO	XHTML element
box	body
groupBox	form

5. Tool support

outputText	text
inputText	input text
radioButton	input radio
checkBox	input checkbox
comboBox with items	select option
button	button

Table 5-2 Mappings between the graphical CIOs and the XHTML elements

5.3 Limitations of current tool support

The current thesis proposes a methodology composed of a set of models gathered in an ontology over which a method manipulating these models is applied thanks to tools that implement it. Our contribution concentrated extensively on the ontological and methodological aspects of the methodology by providing a solution that is independent of the implementation technology. This solution is concretized in a design space that is independent of any implementation language and tool support which represents a contribution to the development process of MM UIs. As a result any MM UIDL could be considered for a possible implementation, while the models and the proposed methodology remain unchanged. Even if these languages didn't currently have the semantical power to support our ontology, they could be extended with new elements in order to reach the required level.

However, with respect to the implementation aspects, this dissertation provides an explicit support concretized in the MultimodalXML tool which applies the described methodology over the ontology implemented in UsiXML language. This technological dependent solution was considered in order to show the feasibility and the proof of concepts without taking into account its usability and performance. For this solution we have identified the following critics:

- Some aspects of the methodology are not supported: only the transformation ensuring the transition from the graphical CUI to the FUI are automated, whereas those from the vocal and MM CUI to their corresponding FUI are ensured manually. A software solution that automates these transformations should be based on the mappings provided in Table 5-1 and Table 5-2.
- When supported, these aspects are not always automated: the transformation rules are manually selected and parameterized by the designer depending on the selected design decision.
- It is not very robust due to the high complexity of issues to be considered: transformation rules are hard to design, implement and apply; in addition, the high dependency between the output produced by one rule and the input manipulated by the next rule to apply determine a very low scalability of the transformational approach.
- It involves a high number of tools (i.e., five tools) which imposes a high threshold i.e.,

5. Tool support

a lot of effort in terms of time and concepts to learn and master before getting familiar with their manipulation.

- The high number of tools to operate with makes them hardly interoperable: any change brought to the UsiXML syntax determines a chain impact over the involved tools as the resultant specification produced by one tool will no longer map the source specification required by the next tool in the chain.

The graphic illustrated in Figure 5-11 shows the position of our solution with respect to the tool complexity and application domain specificity aspects. Currently, most of the tools surveyed in Section 2.4 provide a relative simple solution for problems with a high level of specificity. At the other side of the axis, we find less complex solutions provided by a multiple interoperable intergrated tools that address in exchange more generic aspects of MM UI development. Our approach positions itself somewhere between the two solutions with a high level of complexity but still covering a large spectrum of MM application. The ideal approach is given by the break-even point of the two curbes and consists of a design space-based solution. Its concretization would suppose a single tool (Figure 5-12) that enables designers to: (1) specify the task and domain models of the future system, (2) select for each design option the desired design value that will hide from designers useless details concerning the development steps and sub-steps and the transformation rules supporting them. Before generating the final specification, a preview of the final system could be provided in order to validate the design decision.

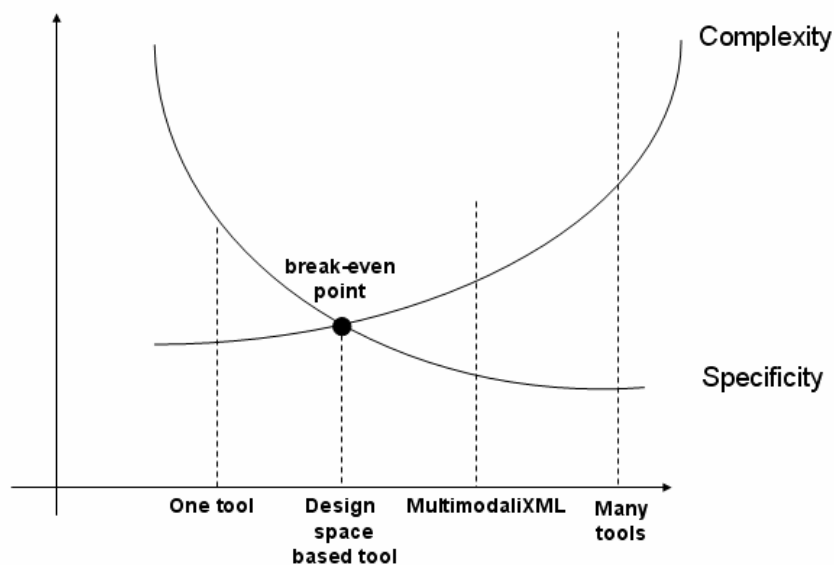


Figure 5-11 Multimodal design tools complexity vs. specificity

5. Tool support

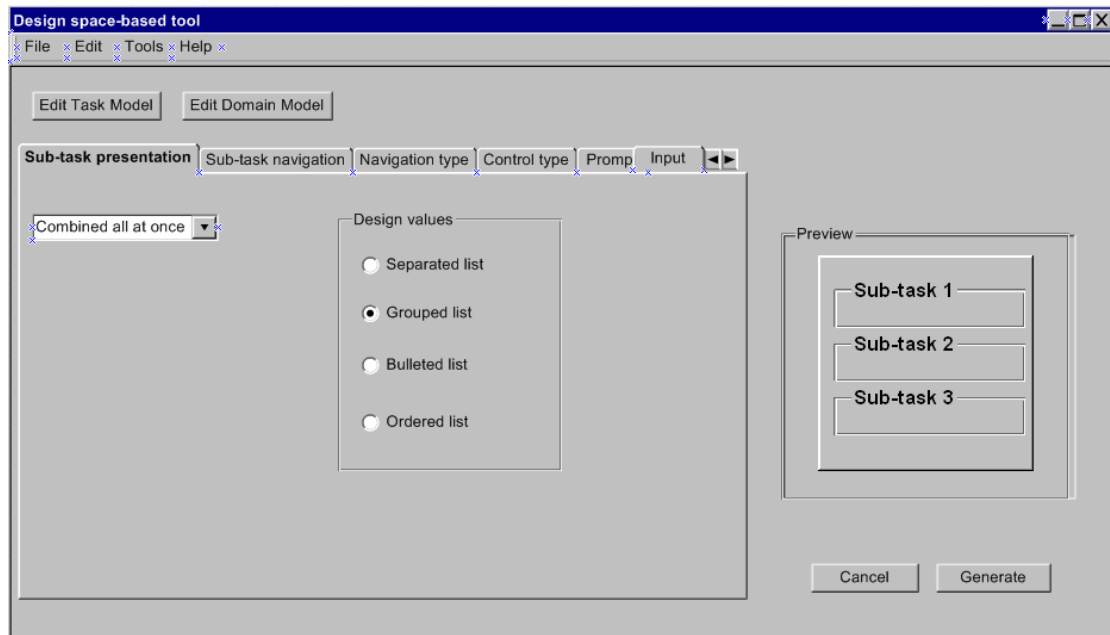


Figure 5-12 A design space-based tool for development of multimodal UIs

5.4 Conclusions

This chapter presented the software solution supporting the method proposed by the current thesis. Each module of the MultimodalXML tool was positioned into the corresponding development step and detailed according to a structured schema. At the end of the chapter several critics of the tool have been identified along with its position among the current MM development tools with respect to the software complexity and application domain specificity. A mock-up of an ideal system supporting the design space was presented as a possible future solution.

6 Validation

6.1 Introduction

After introducing the design space and the transformational approach governing the method applied in the context of this dissertation, the current chapter aims at assessing its validity. We will respond to this issue following two paths:

- *External validation:* based on the software support described in Chapter 5, we show the feasibility of the approach on three case studies having different levels of complexity and coverage. Section 6.2 concerns the development of an on-line polling system, a low complexity web-form application that was selected in order to facilitate the understanding of the proposed method. Section 6.3 details a web-form application of medium complexity dedicated to the development of a car rental system. In Section 6.4 a non-web form application of medium complexity that enables users to browse a map in order to identify different objectives is developed. To solve these case studies we employed the following procedure: (1) Building initial model with their associated tool, (2) Manual editing of transformation rules, where most of them have been elicited prior to realizing these case studies and gathered in the transformation catalog, (3) Manually selecting the transformation rules depending on the design decisions, (4) Automatically applying the selected transformation rules in TransformiXML tool, (5) Transforming the UsiXML specification provided by TransformiXML in the correspondent final UI thanks to the software support presented in Chapter 5.

The validation is supported in Section 6.5 by an empirical study conducted with end users in order to measure the relative usability level provided by different design decisions.

- *Internal validation:* aims at assessing the methodology against the requirements identified at the beginning of this work. For this purpose, Section 6.6 offers a discussion of each requirement based on which an estimation of the methodological coverage is provided.

6.2 Case study 1: Virtual Polling Application

This case study applies our transformational approach for developing a UI on an opinion polling system aiming at collecting opinions of users regarding a certain subject. The scenario of this case study (Figure 6-1) is the following: from the

6. Validation

Task and Domain Models, an AUI is produced, from which three CUIs are derived (GUI, VUI and MM UIs). In the last step, from each CUI a correspondent FUIs is generated.

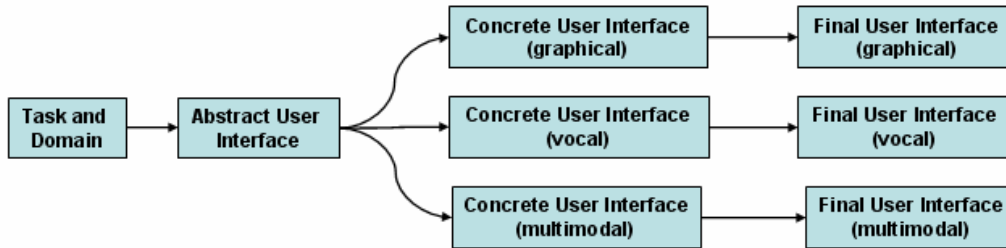


Figure 6-1 Development scenario for virtual polling application

6.2.1 Step 1: The Task and Domain Models

The Task Model, the Domain Model and the mappings between them are graphically described using IdealXML tool. The upper part of Figure 6-2 depicts a CTT representation of the task model envisioned for the future system. The root task consists of participating to an opinion poll. The user has to provide the personal data (i.e., name, zip code, gender, age category). Further, the user iteratively answers some questions as follows: a system task is showing the title of the question and thanks to an interactive task the user is able to select one answer among several proposed ones. Once the questions are answered, the questionnaire is sent back to its initiator. The bottom part of Figure 6-2 illustrates the Domain Model: a participant participates to a questionnaire, a questionnaire is made of several questions and a question is attached to a series of answers.

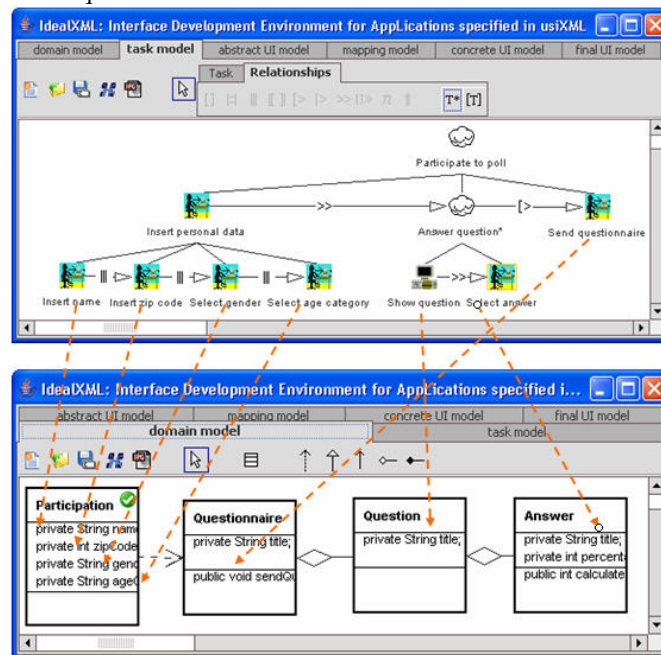


Figure 6-2 Mappings between the Task Model and the Domain Model

6. Validation

The dashed arrows between the two models in Figure 6-2 depict the mappings relationships between the elements of the *Task* and the *Domain Model*. The sub-tasks of *Insert personal data* task is mapped onto the correspondent attributes of *Participation* class (i.e., *name*, *zipCode*, *gender* and *ageCategory*). *Show question* is mapped onto the attribute *title* of class *Question*. The task *Select answer* is mapped onto the attribute *title* of the class *Answer*. Finally, the task *Send questionnaire* is mapped onto the method *sendQuestionnaire* of the class *Questionnaire*. Figure 6-3 illustrates the design of the *Mapping Model* in IdealXML tool. Each leaf task is mapped on the corresponding attribute or method of the classes contained in the *Domain Model*.

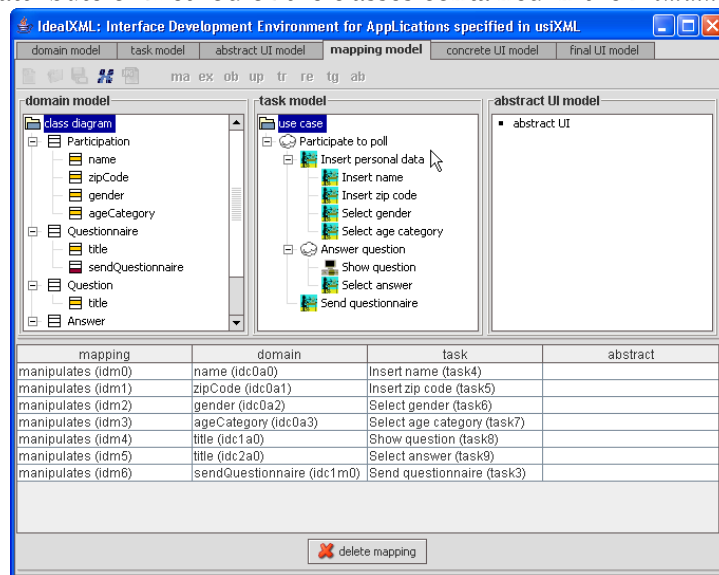


Figure 6-3 Mapping Model for the virtual polling system

IdealXML generates automatically the UsiXML specifications for the *Task Model* (Figure 6-4), *Domain Model* (Figure 6-5) and *Mapping Models* (Figure 6-6).

6. Validation

```
<taskModel id="TaskModelCS1" name="TaskModel">
  <task id="Root" name="Participate to poll" importance="5" category="abstract">
    <task id="T1" name="Insert personal data" importance="3" category="interactive">
      <task id="T11" name="Insert name" taskType="create" taskItem="element" importance="5" category="interactive"/>
      <task id="T12" name="Insert zip code" taskType="create" taskItem="element" importance="5" category="interactive"/>
      <task id="T13" name="Select gender" taskType="select" taskItem="element" importance="5" category="interactive"/>
      <task id="T14" name="Select age category" taskType="select" taskItem="element" importance="5" category="interactive"/>
    </task>
    <task id="T2" name="Answer question" importance="3" category="abstract">
      <task id="T21" name="Show question" taskType="create" taskItem="collection of elements" importance="5" category="system"/>
      <task id="T22" name="Select answer" taskType="select" taskItem="element" importance="5" category="interactive"/>
    </task>
    <task id="T3" name="Send questionnaire" taskType="start" taskItem="operation" importance="3" category="interactive"/>
  </task>
  <enabling id="e1">
    <source sourceId="T1"/>
    <target targetId="T2"/>
  </enabling>
  <disabling id="e2">
    <source sourceId="T2"/>
    <target targetId="T3"/>
  </disabling>
  <independentConcurrency id="e11">
    <source sourceId="T11"/>
    <target targetId="T12"/>
  </independentConcurrency>
  <independentConcurrency id="e12">
    <source sourceId="T12"/>
    <target targetId="T13"/>
  </independentConcurrency>
  <independentConcurrency id="e13">
    <source sourceId="T13"/>
    <target targetId="T14"/>
  </independentConcurrency>
  <iteration id="e3">
    <source sourceId="T2"/>
    <target targetId="T22"/>
  </iteration>
  <enabling id="e21">
    <source sourceId="T21"/>
    <target targetId="T22"/>
  </enabling>
</taskModel>
```

Figure 6-4 Task Model expressed in UsiXML

6. Validation

```

<domainModel id="domainModelCS2" name="domainModel">
  <domainClass id="DC1" name="Participation">
    <attribute id="A1DC1" name="name" attributeDataType="string" attributeCardMin="1" attributeCardMax="1"/>
    <attribute id="A2DC1" name="zipCode" attributeDataType="integer" attributeCardMin="1" attributeCardMax="1"/>
    <attribute id="A3DC1" name="gender" attributeDataType="string" attributeCardMin="1" attributeCardMax="1">
      <enumeratedValue name="Male"/>
      <enumeratedValue name="Female"/>
    </attribute>
    <attribute id="A4DC1" name="ageCategory" attributeDataType="string" attributeCardMin="1" attributeCardMax="1">
      <enumeratedValue name="18-35"/>
      <enumeratedValue name="35-45"/>
      <enumeratedValue name="45+"/>
    </attribute>
  </domainClass>
  <domainClass id="DC2" name="Questionnaire">
    <attribute id="A1DC2" name="title" attributeDataType="string" attributeCardMin="1" attributeCardMax="1"/>
    <method id="M1DC2" name="sendQuestionnaire">
      <param id="P1M1DC1" dataType="Questionnaire" name="qu" paramType="input"/>
    </method>
  </domainClass>
  <domainClass id="DC3" name="Question">
    <attribute id="A1DC3" name="title" attributeDataType="string" attributeCardMin="1" attributeCardMax="1"/>
  </domainClass>
  <domainClass id="DC4" name="Answer">
    <attribute id="A1DC4" name="title" attributeDataType="string" attributeCardMin="1" attributeCardMax="1"/>
    <attribute id="A2DC4" name="percentage" attributeDataType="integer" attributeCardMin="0" attributeCardMax="1"/>
    <method id="M1DC4" name="calculatePercentage">
      <param id="P1M1DC4" dataType="Question" name="qu" paramType="input"/>
      <param id="P2M1DC4" dataType="integer" name="qu" paramType="output"/>
    </method>
  </domainClass>
  <adHoc id="DA1" name="participation" roleACardMin="0" roleACardMax="n" roleBCardMin="0" roleBCardMax="n">
    <source sourceId="DC1"/>
    <target targetId="DC2"/>
  </adHoc>
  <aggregation id="DA2" roleACardMin="1" roleACardMax="n" roleBCardMin="1" roleBCardMax="1">
    <source sourceId="DC2"/>
    <target targetId="DC3"/>
  </aggregation>
  <aggregation id="DA3" roleACardMin="1" roleACardMax="n" roleBCardMin="1" roleBCardMax="1">
    <source sourceId="DC3"/>
    <target targetId="DC4"/>
  </aggregation>
</domainModel>

```

Figure 6-5 Domain Model expressed in UsiXML

```

<mappingModel id="MappingDomainCS2" name="mappingDomain">
  <manipulates id="MA1">
    <source sourceId="T11"/>
    <target targetId="A1DC1"/>
  </manipulates>
  <manipulates id="MA2">
    <source sourceId="T12"/>
    <target targetId="A2DC1"/>
  </manipulates>
  <manipulates id="MA3">
    <source sourceId="T13"/>
    <target targetId="A3DC1"/>
  </manipulates>
  <manipulates id="MA4">
    <source sourceId="T14"/>
    <target targetId="A4DC1"/>
  </manipulates>
  <manipulates id="MA5">
    <source sourceId="T22"/>
    <target targetId="A1DC3"/>
  </manipulates>
  <manipulates id="MA6">
    <source sourceId="T23"/>
    <target targetId="A1DC4"/>
  </manipulates>
  <manipulates id="MA7">
    <source sourceId="T3"/>
    <target targetId="M1DC2"/>
  </manipulates>
</mappingModel>

```

Figure 6-6 Mapping Model expressed in UsiXML

6. Validation

6.2.2 Step 2: From Task and Domain Models to AUI Model

The second transformation step is sub-divided in five sub-steps composed of transformation rules applied in order to realize the transition from the Task and Domain Models to the Abstract Model.

6.2.2.a Sub-step 2.1: Rules for the identification of AUI structure

The current case study considers the following design option values supported by their corresponding rules:

- *Sub-task presentations combined all at once*: Rule 3 and 4
- *Control type with Global containment* (Rule 43) and *Simple cardinality* (Rule 46).

Moreover, Rule 81 is applied in order to create AICs for leaf tasks.

6.2.2.b Sub-step 2.2: Rules for the selection of AICs

The current sub-step generates facets for AICs that support the execution of the leaf task:

- Input facet of type create element for *create name* and *create zipCode* tasks: Rule 83
- Input facet of type select element for *Select gender*, *Select ageCategory* and *Select Answer* tasks: Rule 84. For each enumerated value of an attribute, a selection value with the same name as the enumerated value, will be attached to the above created facet: Rule 85
- Output facet of type convey element for the AIC assigned to the task *Show Question Title*: Rule 86
- As the placement for the control concretization is local Rule 41 is applied in order to generate a control facet of type start operation for the *Send Questionnaire* task.

6.2.2.c Sub-step 2.3: Rules for spatio-temporal arrangement of AIOs

For each couple of sister tasks executed into AIOs, we generate an *abstractAdjacency* relationship between these AIOs. As AIOs can be of two types (i.e., ACs or AICs), there are four possible rules to be applied (Rule 87-90).

6.2.2.d Sub-step 2.4: Rules for the definition of abstract dialog control

By analogy with the previous sub-step, for each couple of sister tasks executed into AIOs, we generate an *abstractDialogControl* relationship between these AIOs that have the same semantics as the temporal relationship defined between the tasks. As AIOs can have two types (i.e., ACs or AICs), there are four possible combination that are considered by Rules 91-94.

6. Validation

6.2.2.e Sub-step 2.5: Rules for the derivation of the AUI to domain mappings

In order to ensure the synchronization between the AICs and attributes of objects from the Domain Model, Rule 95 generates the *updates* relationship. Moreover, Rule 96 enables the triggering of methods by AICs through the *triggers* relationship.

The resultant UsiXML specification issued from the application of the above rules in TransformiXML is illustrated in Figure 6-7.

```
<uiModel name="AUI1" id="AUI1">
  <abstractContainer id="AC1" name="Participate to poll">
    <abstractContainer id="AC11" name="Provide Personal Data">
      <abstractIndividualComponent id="AIC111" name="create name">
        <facet id="FA1111" type="input" name="create name" actionType="create" actionItem="element" dataType="String"/>
      </abstractIndividualComponent>
      <abstractIndividualComponent id="AIC112" name="create ageCategory">
        <facet id="FA1121" type="input" name="create ageCategory" actionType="select" actionItem="element" dataType="String">
          <selectionValue name="18-35"/>
          <selectionValue name="35-45"/>
          <selectionValue name="45+"/>
        </facet>
      </abstractIndividualComponent>
      <abstractIndividualComponent id="AIC113" name="create zipCode">
        <facet id="FA1131" type="input" name="create zipCode" actionType="create" actionItem="element" dataType="String"/>
      </abstractIndividualComponent>
      <abstractIndividualComponent id="AIC114" name="select gender">
        <facet id="FA1141" type="input" name="select gender" actionType="select" actionItem="element" dataType="String">
          <selectionValue name="male"/>
          <selectionValue name="female"/>
        </facet>
      </abstractIndividualComponent>
    </abstractContainer>
    <abstractContainer id="AC12" name="answerQuestionnaire" isRepetitive="true">
      <abstractIndividualComponent id="AIC121" name="output question">
        <facet id="FA1211" type="output" name="output question" actionType="convey" actionItem="element"/>
      </abstractIndividualComponent>
      <abstractIndividualComponent id="AIC122" name="select answer">
        <facet id="FA1221" type="output" name="select answer" actionType="select" actionItem="element" dataType="String"/>
      </abstractIndividualComponent>
    </abstractContainer>
    <abstractIndividualComponent id="AIC13" name="send questionnaire">
      <facet id="FA1221" type="control" name="send questionnaire" actionType="start" actionItem="operation" dataType="String"/>
    </abstractIndividualComponent>
    <uiDialogControl symbol=">>">
      <source sourceId="AIC111"/>
      <target targetId="AIC112"/>
    </uiDialogControl>
    <uiDialogControl symbol=">>">
      <source sourceId="AIC112"/>
      <target targetId="AIC113"/>
    </uiDialogControl>
    <uiDialogControl symbol=">>">
      <source sourceId="AIC113"/>
      <target targetId="AIC114"/>
    </uiDialogControl>
    <uiDialogControl symbol=">>">
      <source sourceId="AC11"/>
      <target targetId="AC12"/>
    </uiDialogControl>
    <uiDialogControl symbol=">>">
      <source sourceId="AIC121"/>
      <target targetId="AIC122"/>
    </uiDialogControl>
    <uiDialogControl symbol=">>">
      <source sourceId="AC12"/>
      <target targetId="AIC13"/>
    </uiDialogControl>
  </abstractContainer>
</uiModel>
```

Figure 6-7 AUI Model expressed in UsiXML

6. Validation

6.2.3 Step 3: From AUI Model to CUI Model

The third step implies a transformational systems composed of transformation rules required to transform the AUI into four different CUIs:

- **Case 1 - graphical CUI:** the modality used to interact with the system is entirely graphical.
- **Case 2 - vocal CUI:** the modality used to interact with the system is entirely vocal.
- **Case 3 - multimodal CUI:** both the graphical and the vocal modalities are employed.

6.2.3.a Case 1: generation of graphical CUI

For the generation of GUIs the designer takes into consideration just the abstract and concrete graphical part of each transformation rule.

6.2.3.a.1 Sub-step 3.1: Reification of AC into CC

For the reification of AC into CC, Rules 15 and 16 are concretizing the separated list design option into graphical objects.

6.2.3.a.2 Sub-step 3.2: Selection of CICs

The current sub-step generates different GICs depending on the type of facets of the corresponding AICs based on the set of design options identified in Section 4.4.3.b.1.2. For each task we specify the considered design option value and the generated CICs:

- Generation of an *outputText* and an *inputText* that enable to *insert the name* and the *zipCode*. Rule 105 is applied each time an AIC with an input facet of type create element is encountered (Table 6-1).

Design option	Value	CIC
Prompting	Graphical (A)	outputText
Input	Graphical (A)	inputText
Immediate feedback	Graphical (A)	inputText
Sub-task guidance	Unguided	-
Answer cardinality	Simple	-
Confirmation answer	Without confirmation	-

Table 6-1 Design option values for inputText

- Generation of a GC of type *box* that will embed a *group of radio buttons* and a GIC of type *outputText* representing the label associated to this group when an input facet of type select element is encountered: Rule 99; The radio buttons associated to this group are created by applying Rule 100. The rules

6. Validation

are used in order to *select the gender* of the user, the *ageCategory* and also his/her *answers* to the questions (Table 6-2).

Design option	Value	CIC
Prompting	Graphical (A)	outputText
Input	Graphical (A)	radioButtons
Immediate feedback	Graphical (A)	radioButtons
Sub-task guidance	Unguided	-
Answer cardinality	Multiple	-
Confirmation answer	Without confirmation	-

Table 6-2 Design option values for radioButtons

- Generation of a GIC of type *outputText*, each time an output facet of type create is encountered: Rule 106 is applied in order to display of the *titles of the questions* (Table 6-3).

Design option	Value	CIC
Prompting	Graphical (A)	outputText

Table 6-3 Design option values for outputText

- Generation of *OK, CANCEL buttons* that will ensure the *Send questionnaire* task and the cancellation of the fulfilled data (Table 6-4): Rule 45.

Design option	Value	CIC
Control type containment	Global	buttons

Table 6-4 Design option values for control buttons

6.2.3.a.3 Sub-step 3.3: Arrangement of CICs

For each couple of adjacent AIOs that are reified into graphicalCICs, we define a graphicalAdjacency relationship between these graphicalCICs. As AIOs can have two types (i.e., ACs or AICs), there are four possible combination to take into account. For each combination a specific rule is considered: Rules 121-124.

6.2.3.a.4 Sub-step 3.4: Navigation definition

The rules that ensure the navigation definition are not applied in the current case study as all the sub-tasks of the virtual polling system are presented combined into the same window.

6.2.3.a.5 Sub-step 3.5: Concrete Dialog Control Definition

For each couple of AIOs with a dialog control relationship, a transposition of this relationship to the graphicalCICs that reify them is realized. As AIOs can have two types (i.e., ACs and AICs), four rules describing the four possible combinations are considered: Rules 125-128.

6. Validation

6.2.3.a.6 Sub-step 3.6: Derivation of CUI to Domain Relationship

Rules 129 and 130 are used to transpose the *updates* and *triggers* relationships from the abstract to the concrete level. These relationships map GICs with attributes and methods from the Domain Model.

6.2.3.b Case 2: generation of vocal UI

For the generation of VUIs the designer takes into consideration just the abstract and concrete vocal part of each transformation rule.

6.2.3.b.1 Sub-step 3.1: Reification of AC into CC

For the reification of AC into CC, Rules 15 and 16 are concretizing the separated list design option into vocal objects.

6.2.3.b.2 Sub-step 3.2: Selection of CICs

The current sub-step generates different VICs depending on the type of facets of the corresponding AICs:

- Generation of a *vocalPrompt*, a *vocalInput* and a *record element* that enable users to utter their *name* (Table 6-5): Rule 105.

Design option	Value	CIC
Prompting	Vocal (A)	vocalPrompt
Input	Vocal (A)	vocalInput + record
Sub-task guidance	Unguided	-
Answer cardinality	Simple	-
Confirmation answer	Without confirmation	-

Table 6-5 Design option values for vocalInput

- Generation of a *vocalPrompt*, a *vocalInput* and a *record* and a VC of type *vocalConfirmation* that enable users to utter and confirm the *zipCode* (Table 6-6): Rule 55.

Design option	Value	CIC
Prompting	Vocal (A)	vocalPrompt
Input	Vocal (A)	vocalInput + record
Sub-task guidance	Unguided	-
Answer cardinality	Simple	-
Confirmation answer	With confirmation	vocalConfirmation

Table 6-6 Design option values for vocalInput with confirmation

- Generation of a *vocalInput*, a *grammar* and the associated *part* element when an input facet of type select is encountered: Rule 99. The rule enables users to specify the *gender*, the *age category* and the *answers to the questions* (Table 6-7).

6. Validation

In order to add the corresponding grammar items for each selection value of the facet, Rule 51 is applied.

Design option	Value	CIC
Prompting	Vocal (A)	vocalPrompt
Input	Vocal (A)	vocalInput
Sub-task guidance	Guided	vocalPrompt
Answer cardinality	Multiple	grammar + part +items
Confirmation answer	Without confirmation	-

Table 6-7 Design option values for vocalInput with grammar items

- Generation of a *vocalPrompt* when an output facet of type convey element is identified (Table 6-8): Rule 106 has to be applied in order to ensure the announcement of the *questionnaire section*.

Design option	Value	CIC
Prompting	Vocal (A)	vocalPrompt

Table 6-8 Design option values for vocalPrompt

- Generation of a *submit* element that enables users to *send the questionnaire* or to *cancel* the fulfilled data (Table 6-9): Rule 45 has to be applied each time a control facet of type start operation is encountered.

Design option	Value	CIC
Control type containment	Global	submit

Table 6-9 Design option values for submit element

6.2.3.b.3 Sub-step 3.3: Arrangement of CICs

For each couple of adjacent AIOs that are reified into vocalCIOs, we define a vocalAdjacency relationship between these vocalCIOs that specify a delay time of 1 second. As vocalCIOs can have two types (i.e., VCs or VICs), there are four possible combination to take into account. For each combination a specific rule is considered: Rule 121-124.

6.2.3.b.4 Sub-step 3.4: Navigation definition

The rules that ensure the navigation definition are not applied in the current case study as all the sub-tasks of the virtual polling system are presented combined into the same window.

6.2.3.b.5 Sub-step 3.5: Concrete Dialog Control Definition

For each couple of AIOs with a dialog control relationship, a transposition of this relationship to the graphicalCIOs that reify them is realized. As AIOs can have two types (i.e., ACs and AICs), four rules describing the four possible combinations are considered: Rules 125-128.

6. Validation

6.2.3.b.6 Sub-step 3.6: Derivation of CUI to Domain Relationship

Rules 129 and 130 are used to transpose the *updates* and *triggers* relationships from the abstract to the concrete level. These relationships map GICs with attributes and methods from the Domain Model.

6.2.3.c Case 3: generation of multimodal UI

For the generation of MMUIs the designer takes into consideration the abstract elements and both the vocal and graphical parts of the transformation rule.

6.2.3.c.1 Sub-step 3.1: Reification of AC into CC

For the reification of AC into CC, Rules 15 and 16 are concretizing the separated list design option into graphical and vocal objects.

6.2.3.c.2 Sub-step 3.2: Selection of CICs

The current sub-step generates different CICs depending on the type of facets of the corresponding AICs and on the design options selected by the designer:

- *Insert name* and *Insert zip code* tasks: Rule 105 has to be applied in order to generate a MM *inputText* widget that supports the following design decisions (Table 6-10).

Design option	Value	CIC
Prompting	Multimodal (R)	outputText + vocalPrompt
Input	Multimodal (E)	inputText + vocalInput + record
Immediate feedback	Graphical (A)	inputText
Guidance for input	Iconic (A)	imageComponents (microphone icon + keyboard icon)
Sub-task guidance	Unguided	-
Answer cardinality	Simple	-
Confirmation answer	Without confirmation	-

Table 6-10 Design option values for multimodal inputText

- *Select gender*, *Select age category* and *Answer to the questions* tasks: Rules 99 and 100 have to be applied in order to generate MM groups of radio buttons (Table 6-11).

Design option	Value	CIC
Prompting	Multimodal (R)	outputText + vocalPrompt
Input	Vocal (A)	vocalInput + grammar + part
Immediate feedback	Graphical (A)	radioButton
Guidance for input	Iconic (A)	imageComponents (microphone

6. Validation

		icon)
Sub-task guidance	Guided	radioButton
Answer cardinality	Simple	-
Confirmation answer	Without confirmation	-

Table 6-11 Design option values for multimodal radioButton

- Generation of *OK*, *CANCEL* buttons that will ensure the *Send questionnaire* task and the cancellation of the fulfilled data (Table 6-12): Rule 45.

Design option	Value	CIC
Control type containment	Global	buttons

Table 6-12 Design option values for outputText

6.2.3.c.3 Sub-step 3.3: Synchronization of CICs

Two rules are used for the synchronization of the previously generated CICs:

- Rule 120 is applied in order to synchronize the *vocalInput* and the *inputText*
- Rule 118 is applied in order to synchronize the *vocalInput* and the GC of type *groupBox* that embeds a set of *radioButtons*

6.2.3.c.4 Sub-step 3.3: Arrangement of CICs

Rules 121-124 are used to specify the arrangement of graphical and vocal CICs.

6.2.3.c.5 Sub-step 3.5: Navigation definition

The rules that ensure the navigation definition are not applied in the current case study as all the sub-tasks of the virtual polling system are presented combined into the same window.

6.2.3.c.6 Sub-step 3.6: Concrete Dialog Control Definition

For each couple of AIOs with a dialog control relationship, a transposition of this relationship to the graphicalCIOs and vocalCIOs that reify them is realized. As AIOs can have two types (i.e., ACs and AICs), four rules describing the four possible combinations are considered: Rules 125-128.

6.2.3.c.7 Sub-step 3.7: Derivation of CUI to Domain Relationship

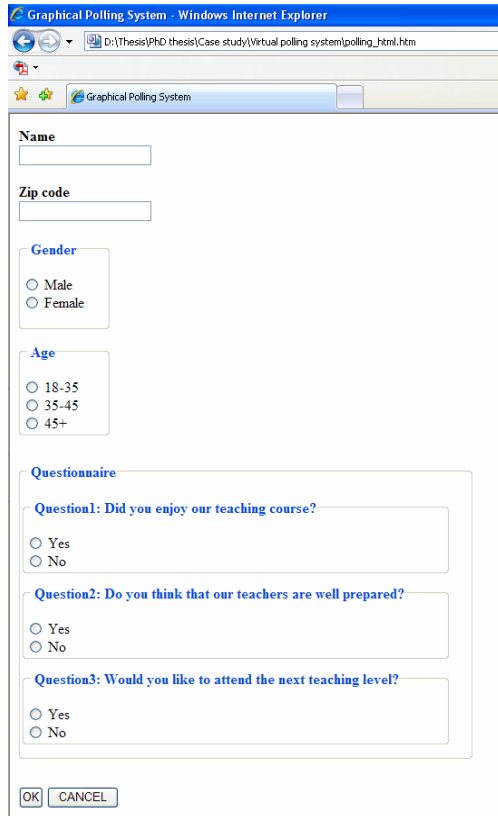
Rules 129 and 130 are used to transpose the *updates* and *triggers* relationships from the abstract to the concrete level. These relationships map GICs and VICs with attributes and methods from the Domain Model.

6.2.4 Step 4: From CUI Model to FUI

This step consists of transforming each variant of the CUI into its respective FUI specification. Hereafter, we illustrate the results of the interpretation of the FUIs

6. Validation

with their corresponding browsers. Thus, Figure 6-8 shows the resultant GUI interpreted with Internet Explorer browser, while Figure 6-9 illustrates a possible User (i.e., U) system (i.e., S) vocal interaction. The MM FUI is interpreted with Opera browser (Figure 6-10).



The screenshot shows a web browser window titled "Graphical Polling System - Windows Internet Explorer". The address bar shows the URL "D:\Thesis\PhD thesis\Case study\Virtual polling system\polling_html.htm". The page content includes a form with the following sections:

- Name:** A text input field.
- Zip code:** A text input field.
- Gender:** Radio buttons for "Male" and "Female".
- Age:** Radio buttons for "18-35", "35-45", and "45+".
- Questionnaire:** Three questions, each with a "Yes" and "No" radio button:
 - Question1: Did you enjoy our teaching course?
 - Question2: Do you think that our teachers are well prepared?
 - Question3: Would you like to attend the next teaching level?

At the bottom of the form are "OK" and "CANCEL" buttons.

Figure 6-8 Graphical UI

S: Welcome to the virtual polling system.
S: Please say your name.
U: John Maverick.
S: What is your zip code?
U: 18.
S: Are you sure?
U: Yes.
S: Please say your gender.
U: Male.
S: Are you sure?
U: Yes.
S: How old are you?
U: 32
S: Are you sure?
U: Yes.
S: Please answer to the following questions
S: Did you enjoy our teaching course?
U: Yes.
S: Do you think that our teachers are well prepared?
U: Yes.
S: Would you like to attend the next teaching level?
U: No.
S: Do you want to send the questionnaire?
U: Yes
S: Are you sure?
U: Yes.

Figure 6-9 Vocal UI

6. Validation

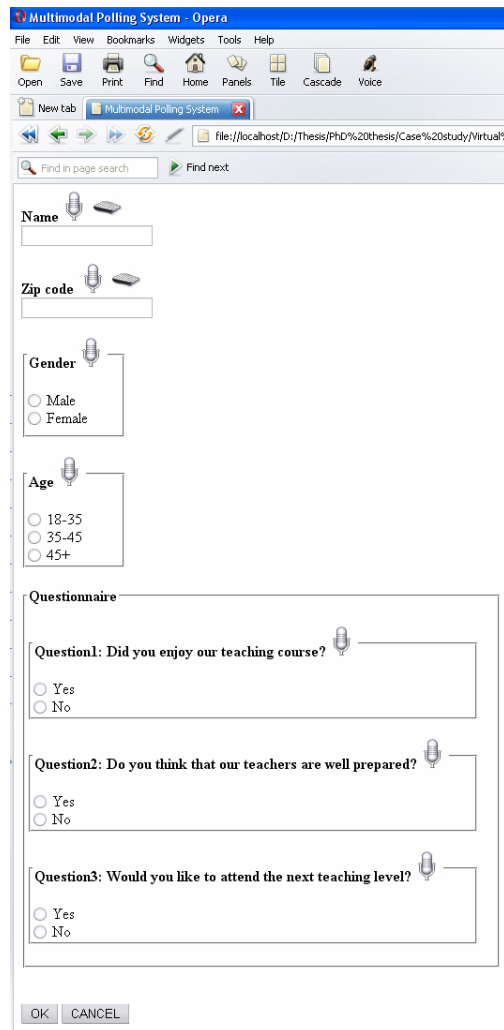


Figure 6-10 Multimodal UI

6.3 Case study 2: Car Rental Application

The second case study is dedicated to an on-line car rental system that allows users to search, select and pay a car based on a set of preferences. The scenario is as follows (Figure 6-11): (1) Task and Domain Models are specified, (2) AUI is generated from these models, (3) three CUIs are derived based on the Input design option values (i.e., graphical, vocal and MM with graphical and vocal equivalence) and (4) three FUIs are derived corresponding to each CUI obtained in the previous step.

6. Validation

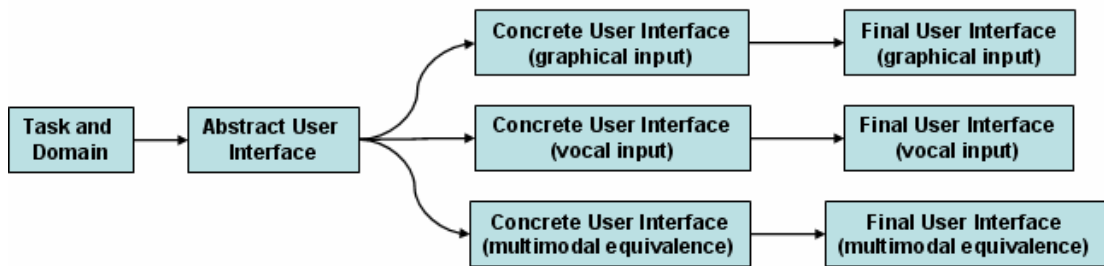


Figure 6-11 Development scenario for car rental application

6.3.1 Step 1: The Task and Domain Models

The root task of the Task Model (Figure 6-12) is decomposed into three basic sub-tasks:

1. **Determine rental preferences** (Figure 6-13): the user has to select a series of information, such as rental location, expected car features, type of insurance. The task is iterative and the user can interrupt it at any moment.
2. **Determine car** (Figure 6-14): the system will launch the search of available cars depending of the preferences established in the previous sub-task. Based on the search results, the user will select the car. The task is iterative and the user can interrupt it at any moment.
3. **Provide payment information** (Figure 6-15): the user provides a set of personal information, such as name and card details. Then, the system checks the validity of the card and finally, the user confirms the payment.

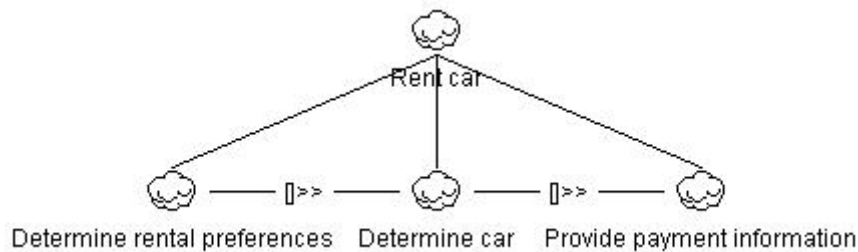


Figure 6-12 The decomposition of Determine rental preferences sub-task

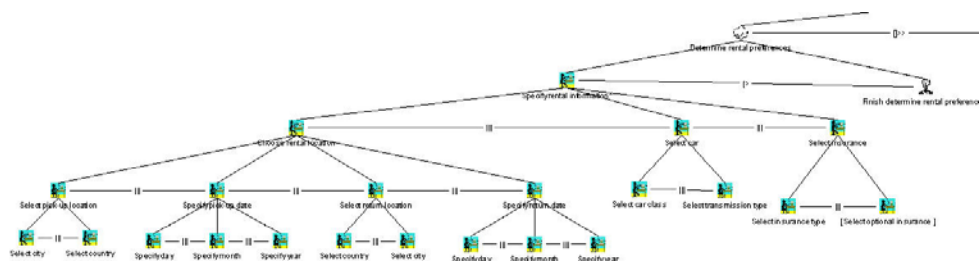


Figure 6-13 The decomposition of Determine rental preferences sub-task

6. Validation

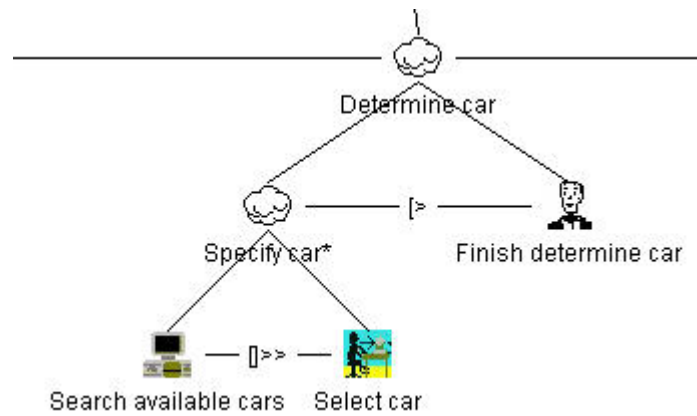


Figure 6-14 The decomposition of Determine car sub-task

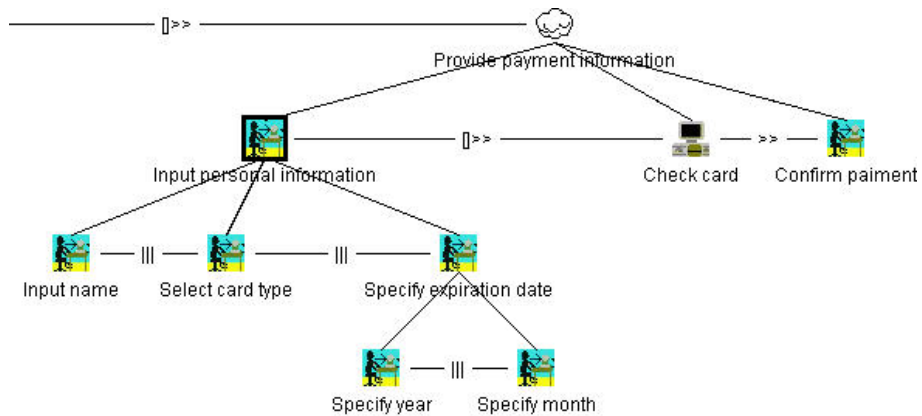


Figure 6-15 The decomposition of Provide payment information sub-task

An excerpt of the UsiXML specification generated in IdealXML corresponding to the *Task Model* is presented Figure 6-16.

6. Validation

```
<taskModel id="TaskModelCS1" name="TaskModel">
  <task id="Root" name="Rent car" importance="5" category="abstract">
    <task id="T1" name="Determine rental preferences" importance="3" category="abstract">
      <task id="T11" name="Specify rental information" importance="3" category="interactive"/>
      <task id="T111" name="Chose rental location" importance="3" category="interactive"/>
      <task id="T1111" name="Select pick-up location" importance="3" category="interactive">
        <task id="T11111" name="Select city" taskType="select" taskItem="element" importance="3" category="interactive"/>
      </task>
      <task id="T1112" name="Select pick-up date" importance="3" category="interactive">
        <task id="T11121" name="Specify day" taskType="select" taskItem="element" importance="3" category="interactive"/>
        <task id="T11122" name="Select month" taskType="select" taskItem="element" importance="3" category="interactive"/>
        <task id="T11123" name="Select year" taskType="select" taskItem="element" importance="3" category="interactive"/>
      </task>
      <task id="T1113" name="Select return location" importance="3" category="interactive">
        <task id="T11131" name="Select city" taskType="select" taskItem="element" importance="3" category="interactive"/>
      </task>
      <task id="T1114" name="Select return date" importance="3" category="interactive">
        <task id="T11141" name="Specify day" taskType="select" taskItem="element" importance="3" category="interactive"/>
        <task id="T11142" name="Select month" taskType="select" taskItem="element" importance="3" category="interactive"/>
        <task id="T11143" name="Select year" taskType="select" taskItem="element" importance="3" category="interactive"/>
      </task>
    </task>
    <task id="T112" name="Select car" importance="3" category="interactive">
      <task id="T1121" name="Select car class" taskType="select" taskItem="element" importance="3" category="interactive"/>
      <task id="T1122" name="Select transimission type" taskType="select" taskItem="element" importance="3" category="interactive"/>
    </task>
    <task id="T113" name="Select insurance" importance="3" category="interactive">
      <task id="T1131" name="Select insurance type" taskType="select" taskItem="element" importance="3" category="interactive"/>
      <task id="T1132" name="Select optional insurance" taskType="select" taskItem="element" importance="3" category="interactive"/>
    </task>
  </task>
  <task id="T12" name="Finish determine rental preferences" taskType="start" taskItem="operation" importance="3" category="user"/>
</task>
<task id="T2" name="Determine car" importance="5" category="abstract">
  <task id="T21" name="Specify car" importance="3" category="abstract">
    <task id="T211" name="Search available cars" taskType="start" taskItem="operation" importance="3" category="system"/>
    <task id="T212" name="Select car" taskType="select" taskItem="element" importance="3" category="interactive"/>
    <task id="T22" name="Finish determine car" taskType="start" taskItem="operation" importance="3" category="user"/>
  </task>
</task>
```

Figure 6-16 Excerpt of Task Model expressed in UsiXML

The *Domain Model* (Figure 6-17) involves 7 classes. *Client* class describes client's characteristics. *Car* class specifies the features of the car, like car class and type of transmission. *Insurance* offers information about the different types of insurances assigned to each car. *RentalInformation* class describes the preferences of the client, such as departure and arrival coordinates, pick up and return dates. *Transaction* class gathers information related to a car rental payment. *CreditCard* provides information about credit cards, the only payment modality available in our system. *Coordinates* is a class used as data type by *RentalInformation* and *Client* classes.

6. Validation

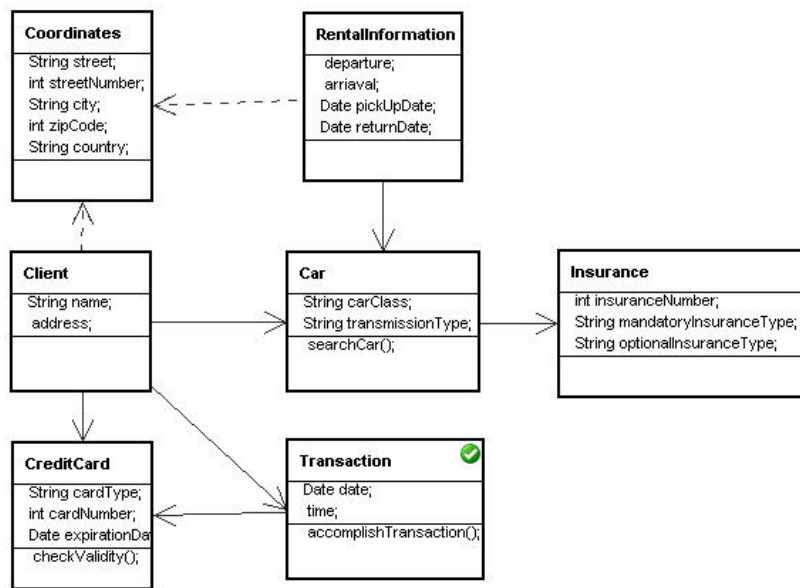


Figure 6-17 Domain Model for the car rental system

Figure 6-18 illustrates an excerpt of the Domain Model expressed in UsiXML language.

```

<domainModel id="domainModelCS2" name="domainModel">
  <domainClass id="DC1" name="Car">
    <attribute id="A1DC1" name="carClass" attributeDataType="string" attributeCardMin="1" attributeCardMax="1">
      <enumeratedValue name="compact"/>
      <enumeratedValue name="mini Van"/>
      <enumeratedValue name="4 Wheel Drive"/>
    </attribute>
    <attribute id="A2DC1" name="transmissionType" attributeDataType="string" attributeCardMin="1" attributeCardMax="1">
      <enumeratedValue name="automatic"/>
      <enumeratedValue name="manual"/>
    </attribute>
  </domainClass>
  <domainClass id="DC2" name="Rental Information">
    <attribute id="A1DC2" name="departure" attributeDataType="Coordinates" attributeCardMin="1" attributeCardMax="1"/>
    <attribute id="A2DC2" name="arrival" attributeDataType="Coordinates" attributeCardMin="1" attributeCardMax="1"/>
    <attribute id="A3DC2" name="pickUpDate" attributeDataType="Date" attributeCardMin="1" attributeCardMax="1"/>
    <attribute id="A4DC2" name="returnDate" attributeDataType="Date" attributeCardMin="1" attributeCardMax="1"/>
  </domainClass>
  <domainClass id="DC3" name="Client">
    <attribute id="A1DC3" name="name" attributeDataType="string" attributeCardMin="1" attributeCardMax="1"/>
    <attribute id="A2DC3" name="address" attributeDataType="Coordinates" attributeCardMin="1" attributeCardMax="1"/>
  </domainClass>
  <domainClass id="DC4" name="Insurance">
    <attribute id="A1DC4" name="insuranceNumber" attributeDataType="integer" attributeCardMin="1" attributeCardMax="1"/>
    <attribute id="A2DC4" name="mandatoryInsuranceType" attributeDataType="string" attributeCardMin="0" attributeCardMax="1">
      <enumeratedValue name="standard"/>
      <enumeratedValue name="full coverage"/>
    </attribute>
    <attribute id="A3DC4" name="optionalInsuranceType" attributeDataType="string" attributeCardMin="0" attributeCardMax="3">
      <enumeratedValue name="loss damage waiver"/>
      <enumeratedValue name="personal accident insurance"/>
      <enumeratedValue name="personal effects protection"/>
    </attribute>
  </domainClass>
  <domainClass id="DC5" name="CreditCard">
    <attribute id="A1DC5" name="cardType" attributeDataType="string" attributeCardMin="1" attributeCardMax="1"/>
    <attribute id="A2DC5" name="cardNumber" attributeDataType="integer" attributeCardMin="1" attributeCardMax="1"/>
    <attribute id="A3DC5" name="expirationDate" attributeDataType="Date" attributeCardMin="1" attributeCardMax="1"/>
    <attribute id="A4DC5" name="checkValidity" attributeDataType="boolean" attributeCardMin="1" attributeCardMax="1"/>
  </domainClass>
  <domainClass id="DC6" name="Transaction">
    <attribute id="A1DC6" name="date" attributeDataType="Date" attributeCardMin="1" attributeCardMax="1"/>
    <attribute id="A2DC6" name="time" attributeDataType="Time" attributeCardMin="1" attributeCardMax="1"/>
    <attribute id="A3DC6" name="accomplishTransaction" attributeDataType="boolean" attributeCardMin="1" attributeCardMax="1"/>
  </domainClass>
</domainModel>

```

Figure 6-18 Excerpts of Domain Model expressed in UsiXML

The mappings between the Task Model and the Domain Model are summed-up in Table 6-13.

6. Validation

Task Model		Domain Model
Select pick-up city	(select element)	RentalInformation.departure.city
Specify day	(select element)	RentalInformation.pickUpDate.day
Specify month	(select element)	RentalInformation.pickUpDate.month
Specify year	(select element)	RentalInformation.pickUpDate.year
Select return city	(select element)	RentalInformation.arrival.city
Specify return day	(select element)	RentalInformation.return.day
Specify return month	(select element)	RentalInformation.return.month
Specify return year	(select element)	RentalInformation.return .year
Select car class	(select element)	Car.carClass
Select transmission type	(select element)	Car.transmissionType
Select insurance type	(select element)	Insurance.mandatoryInsuranceType
Select optional insurance	(select element)	Insurance.optionalInsuranceType
Search available cars	(start operation)	Car.searchCar()
Select car	(select element)	Return parameter of method Car.searchCar()
Input name	(create element)	Client.name
Select card type	(create element)	CreditCard.cardType
Input card number	(create element)	CreditCard.cardNumber
Specify the month of the expiration date	(select element)	CreditCard.expirationDate.month
Specify the year of the expiration date	(select element)	CreditCard.expirationDate.year
Check card	(start operation)	CreditCard.checkValidity()
Confirm payment	(start operation)	Transaction.accomplishTransaction()

Table 6-13 Mappings between task and domain models

6.3.2 Step 2: From Task and Domain Models to AUI Model

The second step considers the generation of the AUI from the previously specified Task and Domain Models

6.3.2.a Sub-step 2.1: Rules for the identification of the AUI structure

The current sub-step considers the following design option values and their corresponding rules:

- *Sub-task presentation in combine grouped lists*: Rule 3 and 4
- *Control concretization with Global placement* (Rules 43) and *Simple cardinality* (Rule 46).

In addition, Rule 81 is applied in order to create AICs for leaf tasks.

6. Validation

6.3.2.b Sub-step 2.2: Rules for the selection of the AICs

The current sub-step generates facets for AICs that support the execution of the leaf task:

- Input facet of type *select element* for the AICs assigned to the following tasks: *select city*, *select day*, *select month*, *select year* for pick-up information as well as for return information, *select car class*, *select transmission type*, *select insurance type*, *select optional insurance*, *select car*, *select expiration date* of the credit card (*month* and *year*): Rule 84; for each enumerated value of the attribute manipulated by the tasks that is executed into the AIC, a selection value with the same name as the enumerated value is attached to the above created facet: Rule 85
- Input facet of type *create element* for the AICs assigned to the *input name* and *input card number* tasks: Rule 83
- Control facets of type *start operation* for the AICs that ensure the data control: Rule 44.

6.3.2.c Sub-step 2.3: Rules for spatio-temporal arrangement of AIOs

For each couple of sister tasks executed into AIOs, we generate an *abstractAdjacency* relationship between these AIOs. As AIOs can have two types (i.e., ACs or AICs), there are four possible rules to be applied (Rule 87-90).

6.3.2.d Sub-step 2.4: Rules for the definition of abstract dialog control

By analogy with the previous sub-step, for each couple of sister tasks executed into AIOs, we generate an *abstractDialogControl* relationship between these AIOs that have the same semantics as the temporal relationship defined between the tasks. As AIOs can have two types (i.e., ACs or AICs), there are four possible combination that are considered by Rules 91-94.

6.3.2.e Sub-step 2.5: Rules for the derivation of the AUI to domain mappings

In order to ensure the synchronization between the AICs and attributes of objects from the Domain Model, Rule 95 generates the *updates* relationship. Moreover, Rule 96 enables the triggering of methods by AICs through the *triggers* relationship.

6.3.3 Step 3: From AUI Model to CUI Model

From the AUI obtained in the previous step, three CUIs will be derived:

- **Case 1 – CUI with graphical input:** the input modality used to interact with the system is entirely graphical.
- **Case 2 - CUI with vocal input:** the input modality used to interact with the system is entirely vocal.

6. Validation

- **Case 3 - CUI with multimodal equivalent input:** graphical or vocal input modalities can be selected to interact with the system.

6.3.3.a Case 1: generation of CUI with graphical input

For this sub-case only the graphical elements of the rules are considered.

6.3.3.a.1 Sub-step 3.1: reification of AC into CC

For the reification of AC into CC, Rules 17 and 18 are concretizing the *combined grouped list* into graphical objects.

6.3.3.a.2 Sub-step 3.2: Selection of CICs

The current sub-step generates different GICs depending on the type of facets of the corresponding AICs and considering the set of design options identified in Section 4.4.3.b.1.2. For each task we specify the considered design option value and the generated CIC:

- For *Input name* and *Input card number* tasks an *outputText* and an *s* are generated (Table 6-14): Rule 105

Design option	Value	CIC
Prompting	Graphical (A)	outputText
Input	Graphical (A)	inputText
Immediate feedback	Graphical (A)	inputText
Sub-task guidance	Unguided	-
Answer cardinality	Simple	inputText
Confirmation answer	Without confirmation	-

Table 6-14 Design option values for inputText

- For each of the following tasks, *Select pick-up and return information (city, day, month, year)*, *Select card type*, *Select expiration date (month, year)*: Rules 97 and 98 generate a *comboBox* widgets (Table 6-15).

Design option	Value	CIC
Prompting	Graphical (A)	outputText
Input	Graphical (A)	comboBox
Immediate feedback	Graphical (A)	comboBox
Sub-task guidance	Guided	items
Answer cardinality	Simple	comboBox
Confirmation answer	Without confirmation	-

Table 6-15 Design option values for comboBox

- For the *Select car class*, *Select transmission type*, *Select insurance type* tasks: Rules 99 and 100 generate *radioButtons* (Table 6-16).

Design option	Value	CIC
Prompting	Graphical (A)	outputText

6. Validation

Input	Graphical (A)	radioButtons
Immediate feedback	Graphical (A)	radioButtons
Sub-task guidance	Guided	radioButtons
Answer cardinality	Simple	radioButtons
Confirmation answer	Without confirmation	-

Table 6-16 Design option values for radioButtons

- For the *Select optional insurance* task: Rules 101 and 102 generate *checkboxes* (Table 6-17).

Design option	Value	CIC
Prompting	Graphical (A)	outputText
Input	Graphical (A)	checkboxes
Immediate feedback	Graphical (A)	checkboxes
Sub-task guidance	Guided	checkboxes
Answer cardinality	Multiple	checkboxes
Confirmation answer	Without confirmation	-

Table 6-17 Design option values for checkboxes

- For the *Select car* task, the graphical elements of Rules 103 and 104 generate a *listBox* widget (Table 6-18).

Design option	Value	CIC
Prompting	Graphical (A)	outputText
Input	Graphical (A)	listBox
Immediate feedback	Graphical (A)	listBox
Sub-task guidance	Guided	items
Answer cardinality	Simple	listBox
Confirmation answer	Without confirmation	-

Table 6-18 Design option values for listBox

- For the *Confirm/cancel payment* task, Rule 45 generates the corresponding buttons.

6.3.3.a.3 Sub-step 3.3: Arrangement of CICs

Rules 121-124 are used to specify the arrangement of graphical CICs.

6.3.3.a.4 Sub-step 3.4: Navigation definition

No navigation is defined as all the graphical components of the present sub-case are presented into the same window and all the vocal components are embedded into the same vocalGroup.

6. Validation

6.3.3.a.5 Sub-step 3.5: Concrete Dialog Control Definition

For each couple of AIOs with a dialog control relationship, a transposition of this relationship to the graphical CIOs that reify them is realized. As AIOs can have two types (i.e., ACs and AICs), four rules describing the four possible combinations are considered: Rules 125-128.

6.3.3.a.6 Sub-step 3.6: Derivation of CUI to Domain Relationship

Rules 129 and 130 are used to transpose the *updates* and *triggers* relationships from the abstract to the concrete level. These relationships map GICs with attributes and methods from the Domain Model.

6.3.3.b Case 2: generation of CUI with vocal input

6.3.3.b.1 Sub-step 3.1: reification of AC into CC

For the reification of AC into CC, Rules 17 and 18 are concretizing the combined grouped list into vocal objects.

6.3.3.b.2 Sub-step 3.2: Selection of CICs

The current sub-step generates different VICs depending on the type of facets of the corresponding AICs based on the set of design options identified in Section 4.4.3.b.1.2. For each task we specify the considered design option value and the generated CIC:

- For *Input name* and *Input card number* tasks (Table 6-19): Rule 60

Design option	Value	CIC
Prompting	Multimodal (R)	outputText + vocalPrompt
Input	Vocal (A)	vocalInput + record
Immediate feedback	Graphical (A)	inputText
Guidance for input	Iconic (A)	imageComponent (microphone icon)
Sub-task guidance	Unguided	-
Answer cardinality	Simple	vocalInput
Confirmation answer	Without confirmation	-

Table 6-19 Design option values for multimodal inputText

- For each of the following tasks, *Select pick-up and return information (city, day, month, year)*, *Select card type*, *Select expiration date (month, year)*, Rules 97 and 98 generate a MM *comboBox* (Table 6-20).

Design option	Value	CIC
Prompting	Multimodal (R)	outputText + vocalPrompt
Input	Vocal (A)	vocalInput + grammar
Immediate feedback	Graphical (A)	comboBox

6. Validation

Guidance for input	Iconic (A)	imageComponent (microphone icon)
Sub-task guidance	Guided	comboBox items+ vocalPrompt + grammar items
Answer cardinality	Simple	comboBox + part
Confirmation answer	Without confirmation	-

Table 6-20 Design option values for multimodal comboBox

- For the *Select car class*, *Select transmission type*, *Select insurance type* tasks: Rules 99 and 100 generate MM *radioButtons* (Table 6-21).

Design option	Value	CIC
Prompting	Multimodal (R)	outputText + vocalPrompt
Input	Vocal (A)	vocalInput + grammar
Immediate feedback	Graphical (A)	radioButtons
Guidance for input	Iconic (A)	imageComponent (microphone icon)
Sub-task guidance	Guided	radioButtons + grammar items
Answer cardinality	Simple	radioButtons + part
Confirmation answer	Without confirmation	-

Table 6-21 Design option values for multimodal radioButtons

- For the *Select optional insurance* task: Rules 101 and 102 generate MM *checkboxes* (Table 6-22).

Design option	Value	CIC
Prompting	Multimodal (R)	outputText + vocalPrompt
Input	Vocal (A)	vocalInput + grammar
Immediate feedback	Graphical (A)	checkboxes
Guidance for input	Iconic (A)	imageComponent (microphone icon)
Sub-task guidance	Guided	checkboxes + grammar items
Answer cardinality	Multiple	checkboxes + part
Confirmation answer	Without confirmation	-

Table 6-22 Design option values for multimodal checkboxes

- For the *Select car* task: Rules 103 and 104 generate a MM *listBox* widgets (Table 6-23).

Design option	Value	CIC
Prompting	Multimodal (R)	outputText + vocalPrompt
Input	Vocal (A)	vocalInput + grammar
Immediate feedback	Graphical (A)	listBox
Guidance for input	Iconic (A)	imageComponent (microphone icon)

6. Validation

		icon)
Sub-task guidance	Guided	listBox items + vocalPrompt + grammar items
Answer cardinality	Simple	listBox + part
Confirmation answer	Without confirmation	-

Table 6-23 Design option values for multimodal listBoxes

- For the *Confirm/cancel payment* task: Rule 45 generates the corresponding *buttons*.

6.3.3.b.3 Sub-step 3.3: Synchronization of CICs

This sub-step is applied in order to ensure the synchronization between vocal CIOs and graphical CIOs generated in the previous sub-step:

- Rule 120 is applied in order to synchronize the *vocalInput* and the *inputText*.
- Rule 116 is applied in order to synchronize the *vocalInput* and the *comboBox*.
- Rule 118 is applied in order to synchronize the *vocalInput* and the *groupBox* that embeds a set of *radioButtons*.
- Rule 117 is applied in order to synchronize the *vocalInput* and the *groupBox* that embeds a set of *checkboxes*.
- Rule 119 is applied in order to synchronize the *vocalInput* and the *listBox*.

6.3.3.b.4 Sub-step 3.3: Arrangement of CICs

Rules 121-124 are used to specify the arrangement of graphical and vocal CICs.

6.3.3.b.5 Sub-step 3.4: Navigation definition

No navigation is defined as all the graphical components of the present sub-case are presented into the same window and all the vocal components are embedded into the same vocalGroup.

6.3.3.b.6 Sub-step 3.5: Concrete Dialog Control Definition

For each couple of AIOs with a dialog control relationship, a transposition of this relationship to the graphical and vocal CIOs that reify them is realized. As AIOs can have two types (i.e., ACs and AICs), four rules describing the four possible combinations are considered: Rules 125-128.

6.3.3.b.7 Sub-step 3.6: Derivation of CUI to Domain Relationship

Rules 129 and 130 are used to transpose the *updates* and *triggers* relationships from the abstract to the concrete level. These relationships map GICs and VICs with attributes and methods from the Domain Model.

6. Validation

6.3.3.c Case 3 : generation CUI with multimodal equivalent input

The current case contains transformation rules applied on the AUI produced in the previous step, in order to generate the correspondent MM CUI with equivalent graphical and vocal input.

6.3.3.c.1 Sub-step 3.1: reification of AC into CC

For the reification of AC into CC, Rules 17 and 18 are concretizing the combined grouped list into graphical and vocal objects.

6.3.3.c.2 Sub-step 3.2: Selection of CICs

The current sub-step generates different GICs and VICs depending on the type of facets of the corresponding AICs based on the set of design option identified in Section 4.4.3.b.1.2. For each task we specify the considered design option value and the generated CICs:

- For *Input name* and *Input card number* tasks a MM *inputText* is generated (Table 6-24): Rule 105.

Design option	Value	CIC
Prompting	Multimodal (R)	outputText+vocalPrompt
Input	Multimodal (E)	inputText + vocalInput + record
Immediate feedback	Graphical (A)	inputText
Guidance for input	Iconic (A)	imageComponents (microphone + keyboard icons)
Sub-task guidance	Unguided	-
Answer cardinality	Simple	inputText + vocalInput
Confirmation answer	Without confirmation	-

Table 6-24 Design option values for multimodal inputText

- For each of the following tasks, *Select pick-up and return information (city, day, month, year)*, *Select card type*, *Select expiration date (month, year)*: Rule 97 and 98 generate a *comboBox* (Table 6-25).

Design option	Value	CIC
Prompting	Multimodal (R)	outputText + vocalPrompt
Input	Multimodal (E)	comboBox + vocalInput + grammar
Immediate feedback	Graphical (A)	comboBox
Guidance for input	Iconic (A)	imageComponents (microphone + mouse icons)
Sub-task guidance	Guided	comboBox items+ vocalPrompt + grammar items
Answer cardinality	Simple	comboBox + part

6. Validation

Confirmation answer	Without confirmation	-
---------------------	----------------------	---

Table 6-25 Design option values for multimodal combobox

- For the *Select car class*, *Select transmission type*, *Select insurance type* tasks: Rules 99 and 100 generate MM *radioButtons* (Table 6-26).

Design option	Value	CIC
Prompting	Multimodal (R)	outputText + vocalPrompt
Input	Multimodal (E)	radioButtons + vocalInput + grammar
Immediate feedback	Graphical (A)	radioButtons
Guidance for input	Iconic (A)	imageComponent (microphone + keyboard icons)
Sub-task guidance	Guided	radioButtons + grammar items
Answer cardinality	Simple	radioButtons + part
Confirmation answer	Without confirmation	-

Table 6-26 Design option values for multimodal radioButtons

- For the *Select optional insurance* task: Rules 101 and 102 generate *checkboxes* (Table 6-27).

Design option	Value	CIC
Prompting	Multimodal (R)	outputText + vocalPrompt
Input	Multimodal (E)	checkboxes + vocalInput + grammar
Immediate feedback	Graphical (A)	checkboxes
Guidance for input	Iconic (A)	imageComponent (microphone icon)
Sub-task guidance	Guided	checkboxes + grammar items
Answer cardinality	Multiple	checkboxes + part
Confirmation answer	Without confirmation	-

Table 6-27 Design option values for multimodal checkboxes

- For the *Select car* task: Rules 103 and 104 generate a MM *listBox* widgets (Table 6-28).

Design option	Value	CIC
Prompting	Multimodal (R)	outputText + vocalPrompt
Input	Multimodal (E)	listBox + vocalInput + grammar
Immediate feedback	Graphical (A)	listBox
Guidance for input	Iconic (A)	imageComponents (microphone+keyboard icons)
Sub-task guidance	Guided	listBox items + vocalPrompt + grammar items
Answer cardinality	Simple	listBox + part

6. Validation

Confirmation answer	Without confirmation	-
---------------------	----------------------	---

Table 6-28 Design option values for multimodal listBoxes

- For the *Confirm/cancel payment* task, Rule 45 generates the corresponding buttons and submit elements.

6.3.3.c.3 Sub-step 3.3: Synchronization of CICs

The rules identified in the correspond section of the previous case are reused in order to ensure this sub-step.

6.3.3.c.4 Sub-step 3.4: Arrangement of CICs

Rules 121-124 are used to specify the arrangement of graphical and vocal CICs.

6.3.3.c.5 Sub-step 3.5: Navigation definition

No navigation is defined as all the graphical components of the present sub-case are presented into the same *window* and all the vocal components are embedded into the same *vocalGroup*.

6.3.3.c.6 Sub-step 3.6: Concrete Dialog Control Definition

For each couple of AIOs with a dialog control relationship, a transposition of this relationship to the graphical and vocal CIOs that reify them is realized. As AIOs can have two types (i.e., ACs and AICs), four rules describing the four possible combinations are considered: Rules 125-128.

6.3.3.c.7 Sub-step 3.7: Derivation of CUI to Domain Relationship

Rules 129 and 130 are used to transpose the *updates* and *triggers* relationships from the abstract to the concrete level. These relationships map GICs and VICs with attributes and methods from the Domain Model.

6.3.4 Step 4: From CUI Model to FUI

This step consists of transforming each version of the CUI into its corresponding FUI specification. The resultant FUIs interpreted with Opera browser are: FUI enabling graphical input (Figure 6-19), FUI enabling vocal input (Figure 6-20) and the FUI enabling equivalent graphical and vocal input (Figure 6-21).

6. Validation

The screenshot shows a graphical user interface for a car rental system. The window title is "Car Rental System". The interface is divided into several sections:

- Rental preferences:**
 - Pick-up information:** City (dropdown), Pick-up date (Day, Month, Year dropdowns).
 - Return information:** City (dropdown), Return date (Day, Month, Year dropdowns).
 - Car information:** Transmission type (radio buttons for automatic/manual), Car class (radio buttons for compact/economy/sport).
 - Insurance information:** Insurance type (radio buttons for standard/full coverage), Optional insurance (checkboxes for juridic protection, personal accident, personal effects).
- Search available cars:** A button with a microphone icon.
- Car selection:** A box with a "Select a car" button.
- Payment information:** Name (text input), Card type (dropdown), Card number (text input), Expiration date (Month, Year dropdowns).
- Buttons:** OK and CANCEL buttons at the bottom.

Figure 6-19 FUI – graphical input

The screenshot shows the same graphical user interface as Figure 6-19, but with vocal input capabilities. The window title is "Car Rental System". The interface is identical to Figure 6-19, but with microphone icons next to the following fields:

- City (dropdown)
- Pick-up date (Day, Month, Year dropdowns)
- Return date (Day, Month, Year dropdowns)
- Transmission type (radio buttons)
- Car class (radio buttons)
- Insurance type (radio buttons)
- Optional insurance (checkboxes)
- Name (text input)
- Card type (dropdown)
- Card number (text input)
- Expiration date (Month, Year dropdowns)

The "Search available cars" button also has a microphone icon. The "OK" and "CANCEL" buttons are at the bottom.

Figure 6-20 FUI – vocal input

6. Validation

The screenshot displays a web browser window with a URL that includes 'Multimodal%20Car%20rental%20sys'. The page content is titled 'Rental preferences' and is organized into several sections, each with a microphone icon indicating voice input capability:

- Pick-up information:** Includes a 'City' dropdown menu, a 'Pick-up date' section with 'Day' (1), 'Month' (January), and 'Year' (2007) dropdowns.
- Return information:** Includes a 'City' dropdown menu, a 'Return date' section with 'Day' (1), 'Month' (January), and 'Year' (2007) dropdowns.
- Car information:** Includes 'Transmission type' with radio buttons for 'automatic' and 'manual', and 'Car class' with radio buttons for 'compact', 'economy', and 'sport'.
- Insurance information:** Includes 'Insurance type' with radio buttons for 'standard' and 'full coverage', and 'Optional insurance' with checkboxes for 'juridic protection', 'personal accident', and 'personal effects'.

Below these sections are two buttons: 'Search available cars' and 'Car selection' (with a 'Select a car' button inside a box). At the bottom is a 'Payment information' section with fields for 'Name', 'Card type' (set to VISA), 'Card number', and 'Expiration date' (with 'Month' and 'Year' dropdowns). At the very bottom are 'OK' and 'CANCEL' buttons.

Figure 6-21 FUI – equivalent graphical and vocal input

6.4 Case study 3: Map Browsing Application

The third case study considers a non web-form application that allows users to browse a map in order to identify different objectives. The scenario is as follows

6. Validation

(Figure 6-22): (1) Task and Domain Models are specified, (2) AUI is generated from these models, (3) three CUIs are derived based on the *Input design option* values (i.e., graphical, vocal and MM) and (4) three FUIs are derived corresponding to each CUI obtained in the previous step.

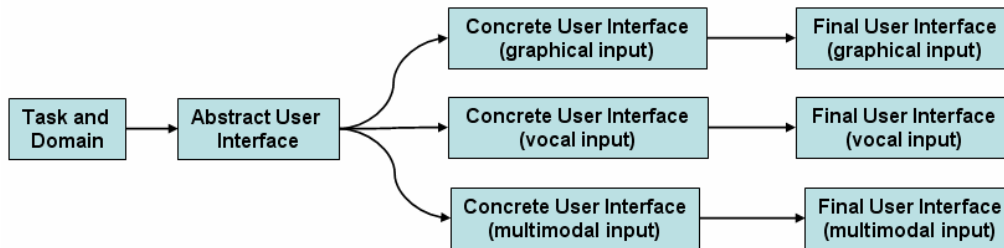


Figure 6-22 Development scenario for the map browsing system

6.4.1 Step 1: The Task and Domain Models

The task (Figure 6-23) consisted in browsing a 3 by 3 grid map for which a guidance with respect to the structure of the browsing instruction was provided [Stan07]. This structure was obtained thanks to the support offered by our ontology to the general structure of an instruction (Section 3.3.3) which enabled us to specify its components:

- Action: translate, zoom in, zoom out.
- Object: image displaying the map.
- Parameter X: left, center, right.
- Parameter Y: top, center, bottom.

Due to the fact that there is only one map to manipulate, the object became non-mandatory when specifying the instruction. The selection of the action has to be followed by the specification of the two parameters which were aggregated in order to provide an easier specification of the browsing direction (e.g., top left, top right, bottom left). Once the instruction is conveyed the system is updating the image corresponding to the specified instruction.

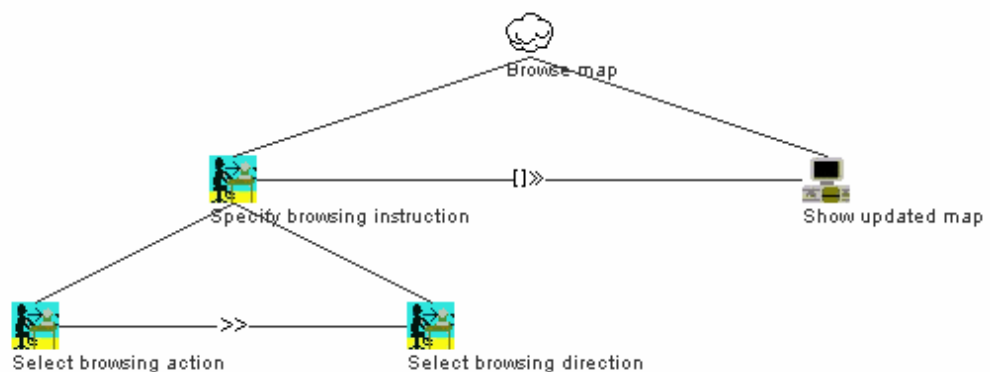


Figure 6-23 Task Model of the map browsing application

6. Validation

The UsiXML specification corresponding to the *Task Model* generated in IdealXML is presented in Figure 6-24.

```
<taskModel id="TaskModelCS1" name="TaskModel">
  <task id="Root" name="Browse map" importance="5" category="abstract">
    <task id="T1" name="Specify instruction" importance="3" category="interactive">
      <task id="T11" name="Select browsing action" taskType="select" taskItem="element" importance="3" category="interactive"/>
      <task id="T12" name="Select browsing direction" taskType="select" taskItem="element" importance="3" category="user"/>
    </task>
    <task id="T2" name="Show updated map" importance="5" taskType="convey" taskItem="element" category="system"/>
  </task>
  <enablingWithInformationPasing id="e1">
    <source sourceId="T1"/>
    <target targetId="T2"/>
  </enablingWithInformationPasing>
  <enabling id="e2">
    <source sourceId="T11"/>
    <target targetId="T12"/>
  </enabling>
</taskModel>
```

Figure 6-24 Specification of the Task Model in UsiXML

The Domain Model (Figure 6-25) involves four classes: (1) the *Image* class specifies the features of the images that can be browsed by the user, (2) the *VisibilityZone* class determines the area of the image that is visible for the users, (3) the *ExplorationZone* class defines the complete area including the non-visible as well as the visible part of the image and (4) the *Cell* class determines the positions of the cells composing the exploration and the visibility zones of the image.

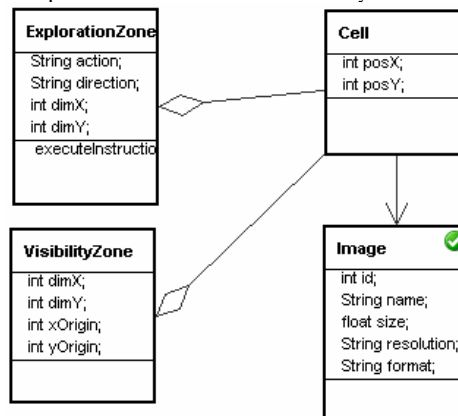


Figure 6-25 Domain Model for the map browsing system

Figure 6-26 illustrates an excerpt of the Domain Model expressed in UsiXML.

6. Validation

```

<domainModel id="domainModelCS2" name="domainModel">
  <domainClass id="DC1" name="ExplorationZone">
    <attribute id="A1DC1" name="action" attributeDataType="string" attributeCardMin="1" attributeCardMax="1">
      <enumeratedValue name="translate"/>
      <enumeratedValue name="zoom in"/>
      <enumeratedValue name="zoom out"/>
    </attribute>
    <attribute id="A2DC1" name="direction" attributeDataType="string" attributeCardMin="1" attributeCardMax="1">
      <enumeratedValue name="top left"/>
      <enumeratedValue name="top"/>
      <enumeratedValue name="top right"/>
      <enumeratedValue name="left"/>
      <enumeratedValue name="center"/>
      <enumeratedValue name="right"/>
      <enumeratedValue name="bottom left"/>
      <enumeratedValue name="bottom right"/>
    </attribute>
    .....
    <method id="M1DC6" name="executeInstruction">
      <param id="P1M1DC1" dataType="action" name="action_param" paramType="input"/>
      <param id="P2M1DC1" dataType="direction" name="direction_param" paramType="input"/>
    </method>
  </domainClass>
  <domainClass id="DC2" name="visibilityZone">
    <attribute id="A1DC2" name="dimX" attributeDataType="integer" attributeCardMin="1" attributeCardMax="1"/>
    <attribute id="A2DC2" name="dimY" attributeDataType="integer" attributeCardMin="1" attributeCardMax="1"/>
    <attribute id="A3DC2" name="xOrigin" attributeDataType="integer" attributeCardMin="1" attributeCardMax="1"/>
    <attribute id="A4DC2" name="yOrigin" attributeDataType="integer" attributeCardMin="1" attributeCardMax="1"/>
  </domainClass>
  <domainClass id="DC2" name="cell">
    <attribute id="A1DC2" name="posX" attributeDataType="integer" attributeCardMin="1" attributeCardMax="1"/>
    <attribute id="A2DC2" name="posY" attributeDataType="integer" attributeCardMin="1" attributeCardMax="1"/>
  </domainClass>
  <domainClass id="DC2" name="image">
    <attribute id="A1DC2" name="id" attributeDataType="integer" attributeCardMin="1" attributeCardMax="1"/>
    <attribute id="A2DC2" name="name" attributeDataType="string" attributeCardMin="1" attributeCardMax="1"/>
    <attribute id="A3DC2" name="size" attributeDataType="float" attributeCardMin="1" attributeCardMax="1"/>
    <attribute id="A4DC2" name="resolution" attributeDataType="string" attributeCardMin="1" attributeCardMax="1"/>
    <attribute id="A4DC2" name="format" attributeDataType="string" attributeCardMin="1" attributeCardMax="1"/>
  </domainClass>
  .....
</domainModel>

```

Figure 6-26 Excerpts of Domain Model expressed in UsiXML

The mappings between the Task Model and the Domain Model are summed-up in Table 6-29.

Task Model	Domain Model
Select browsing action (select element)	ExplorationZone.action
Select browsing direction (select element)	ExplorationZone.direction
Show updated image (convey element)	ExplorationZone.executeInstruction()

Table 6-29 Mappings between task and domain models

6.4.2 Step 2: From Task and Domain Models to AUI Model

The second step considers the generation of the AUI from the previously specified Task and Domain Models.

6.4.2.a Sub-step 2.1: Rules for the identification of the AUI structure

The current sub-step considers the *Sub-task presentation* design option that is conveyed in *combined grouped lists*: Rule 3 and 4. Moreover, Rule 81 is applied in order to create AICs for leaf tasks.

6. Validation

6.4.2.b Sub-step 2.2: Rules for the selection of the AICs

The current sub-step generates facets for AICs that support the execution of the leaf task.

- Input facet of type *select element* for the AICs assigned to the following tasks: *Select browsing action*, *select browsing direction*: Rule 84; for each enumerated value of the attribute manipulated by the tasks that is executed into the AIC, a selection value with the same name as the enumerated value is attached to the above created facet: Rule 85
- Output facet of type *convey element* for the AIC assigned to the *Show updated map* task: Rule 82.

6.4.2.c Sub-step 2.3: Rules for spatio-temporal arrangement of AIOs

For each couple of sister tasks executed into AIOs, we generate an *abstractAdjacency* relationship between these AIOs. As AIOs can have two types (i.e., ACs or AICs), there are four possible rules to be applied (Rules 87-90).

6.4.2.d Sub-step 2.4: Rules for the definition of abstract dialog control

For each couple of sister tasks executed into AIOs, we generate an *abstractDialogControl* relationship between these AIOs that have the same semantics as the temporal relationship defined between the tasks. As AIOs can have two types (i.e., ACs or AICs), there are four possible combination that are considered by Rules 91-94.

6.4.2.d.1 Sub-step 2.5: Rules for the derivation of the AUI to domain mappings

In order to ensure the synchronization between the AICs and attributes of objects from the Domain Model, Rule 95 generates the *updates* relationship. Moreover, Rule 96 enables the triggering of methods by AICs through the *triggers* relationship.

6.4.3 Step 3: From AUI Model to CUI Model

For the AUI obtained in the previous step, three CUI will be derived:

- **Case 1 – CUI with graphical input:** the input modality used to specify the instruction is entirely graphical.
- **Case 2 - CUI with vocal input:** the input modality used to specify the instruction is entirely vocal.
- **Case 3 - CUI with multimodal input:** for the specification of the action the graphical modality is assigned, whereas for the direction the vocal assignement was considered.

6. Validation

6.4.3.a Case 1: generation of CUI with graphical input

The current case contains transformation rules applied on the AUI produced in the previous step, in order to generate the correspondent graphical CUI with graphical assignement for both the browsing action the browsing direction.

6.4.3.a.1 Sub-step 3.1: reification of AC into CC

For the reification of AC into CC, Rules 17 and 18 are concretizing the *combined grouped list* into graphical objects.

6.4.3.a.2 Sub-step 3.2: Selection of CICs

The current sub-step generates different GICs depending on the type of facets of the corresponding AICs and considering the set of design options identified in Section 4.4.3.b.1.2. For each task we specify the considered design option value and the generated CIC:

- For the *Select action* task: Rules 99 and 100 generate *radioButtons* (Table 6-30).

Design option	Value	CIC
Prompting	Graphical (A)	outputText
Input	Graphical (A)	radioButtons
Immediate feedback	Graphical (A)	radioButtons
Guidance for input	Iconic (A)	imageComponent (mouse icon)
Sub-task guidance	Guided	radioButtons
Answer cardinality	Simple	radioButtons
Confirmation answer	Without confirmation	-

Table 6-30 Design option values for multimodal radioButtons

- For the *Select direction* task: Rules 109 and 110 generate *imageZones* embedded in *imageComponents* (Table 6-31).

Design option	Value	CIC
Prompting	Graphical (A)	outputText
Input	Graphical (A)	imageComponent
Guidance for input	Iconic (A)	imageComponent (mouse icon)
Sub-task guidance	Guided	imageZones
Answer cardinality	Simple	radioButtons
Confirmation answer	Without confirmation	-

Table 6-31 Design option values for imageZones

- For *display the updated image* task: Rule 115 generates an *imageComponent*.

6.4.3.a.3 Sub-step 3.3: Arrangement of CICs

Rules 121-124 are used to specify the arrangement of graphical CICs.

6. Validation

6.4.3.a.4 Sub-step 3.4: Navigation definition

No navigation is defined as all the graphical components of the present sub-case are presented into the same window.

6.4.3.a.5 Sub-step 3.5: Concrete Dialog Control Definition

For each couple of AIOs with a dialog control relationship, a transposition of this relationship to the graphical CIOs that reify them is realized. As AIOs can have two types (i.e., ACs and AICs), four rules describing the four possible combinations are considered: Rules 125-128.

6.4.3.a.6 Sub-step 3.6: Derivation of CUI to Domain Relationship

Rules 129 and 130 are used to transpose the *updates* and *triggers* relationships from the abstract to the concrete level. These relationships map GICs with attributes and methods from the Domain Model.

6.4.3.b Case 2 - CUI with vocal input

The current case contains transformation rules applied on the AUI produced in the previous step, in order to generate the correspondent MM CUI with vocal assignement for both the browsing action browsing direction.

6.4.3.b.1 Sub-step 3.1: reification of AC into CC

For the reification of AC into CC, Rules 17 and 18 are concretizing the *combined grouped list* into graphical and vocal objects.

6.4.3.b.2 Sub-step 3.2: Selection of CICs

The current sub-step generates different GICs and VICs depending on the type of facets of the corresponding AICs and considering the set of design options identified in Section 4.4.3.b.1.2. The *select action* and the *select direction* tasks are expressed in one single utterance, therefore the Rules 111 and 112 have to be applied (Table 6-32).

Design option	Value	CIC
Prompting	Graphical (A)	outputText
Input	Vocal (A)	vocalInput + grammar + part
Guidance for input	Iconic (A)	imageComponent (microphone)
Sub-task guidance	Guided	outputText+ radioButton
Answer cardinality	Simple	outputText + radioButton
Confirmation answer	Without confirmation	-

Table 6-32 Design option values for multimodal radioButton

- For *display the updated image* task: Rule 115 generates an *imageComponent*.

6. Validation

6.4.3.b.3 Sub-step 3.3: Arrangement of CICs

Rules 121-124 are used to specify the arrangement of graphical and vocal CICs.

6.4.3.b.4 Sub-step 3.4: Navigation definition

No navigation is defined as all the graphical components of the present sub-case are presented into the same *window* and all the vocal components are embedded into the same *vocalGroup*.

6.4.3.b.5 Sub-step 3.5: Concrete Dialog Control Definition

For each couple of AIOs with a dialog control relationship, a transposition of this relationship to the graphical and vocal CIOs that reify them is realized. As AIOs can have two types (i.e., ACs and AICs), four rules describing the four possible combinations are considered: Rules 125-128.

6.4.3.b.6 Sub-step 3.6: Derivation of CUI to Domain Relationship

Rules 129 and 130 are used to transpose the *updates* and *triggers* relationships from the abstract to the concrete level. These relationships map GICs and VICs with attributes and methods from the Domain Model.

6.4.3.c Case 3 - CUI with multimodal input

The current case contains transformation rules applied on the AUI produced in the previous step, in order to generate the correspondent MM CUI with graphical assignement for the browsing action and vocal assignement for the browsing direction.

6.4.3.c.1 Sub-step 3.1: reification of AC into CC

For the reification of AC into CC, Rules 17 and 18 are concretizing the combined grouped list into graphical and vocal objects.

6.4.3.c.2 Sub-step 3.2: Selection of CICs

The current sub-step generates different GICs depending on the type of facets of the corresponding AICs and considering the set of design options identified in Section 4.4.3.b.1.2. For each task we specify the considered design option value and the generated CIC:

- For the *Select action* task: Rules 99 and 100 generate MM *radioButtons* (Table 6-33).

Design option	Value	CIC
Prompting	Graphical (A)	outputText
Input	Graphical (A)	radioButtons
Immediate feedback	Graphical (A)	radioButtons
Guidance for input	Iconic (A)	imageComponent (mouse icon)

6. Validation

Sub-task guidance	Guided	radioButtons
Answer cardinality	Simple	radioButtons
Confirmation answer	Without confirmation	-

Table 6-33 Design option values for multimodal radioButtons

- For the *Select direction* task: Rules 113 and 114 generate *imageZones* embedded in *imageComponents* and *grammar items* (Table 6-34).

Design option	Value	CIC
Prompting	Graphical (A)	outputText
Input	Vocal (A)	vocalInput + grammar
Guidance for input	Iconic (A)	imageComponent (microphone icon)
Sub-task guidance	Guided	imageZones + items
Answer cardinality	Simple	imageZones + part
Confirmation answer	Without confirmation	-

Table 6-34 Design option values for multimodal radioButtons

- For *display the updated image* task, Rule 129 generates an *imageComponent*.

6.4.3.c.3 Sub-step 3.3: Arrangement of CICs

Rules 113-116 are used to specify the arrangement of graphical and vocal CICs.

6.4.3.c.4 Sub-step 3.4: Navigation definition

No navigation is defined as all the graphical components of the present sub-case are presented into the same *window* and all the vocal components are embedded into the same *vocalGroup*.

6.4.3.c.5 Sub-step 3.5: Concrete Dialog Control Definition

For each couple of AIOs with a dialog control relationship, a transposition of this relationship to the graphical and vocal CIOs that reify them is realized. As AIOs can have two types (i.e., ACs and AICs), four rules describing the four possible combinations are considered: Rules 125-128.

6.4.3.c.6 Sub-step 3.6: Derivation of CUI to Domain Relationship

Rules 129 and 130 are used to transpose the *updates* and *triggers* relationships from the abstract to the concrete level. These relationships map GICs and VICs with attributes and methods from the Domain Model.

6. Validation

6.4.4 Step 4: From CUI Model to FUI

This step consists of transforming each variant of the CUI into its respective FUI specification. The resultant FUI interpreted with Opera browser is the following: FUI enabling graphical input (Figure 6-27), FUI enabling vocal input (Figure 6-28), FUI enabling equivalent graphical input for specifying browsing action and and vocal input for the browsing direction (Figure 6-29).

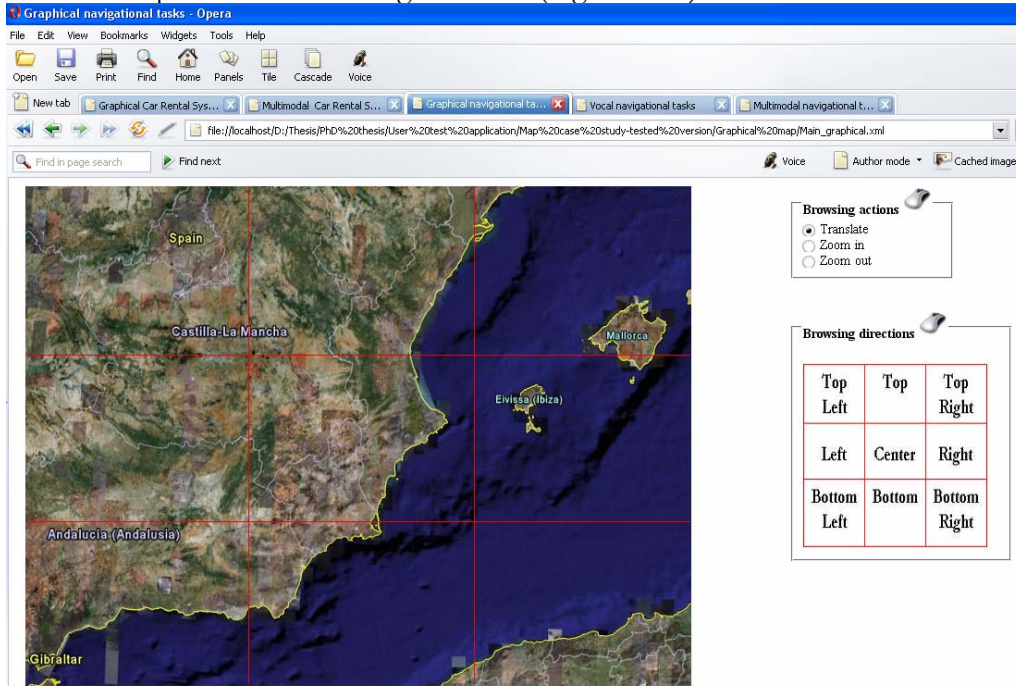


Figure 6-27 FUI – graphical input

6. Validation

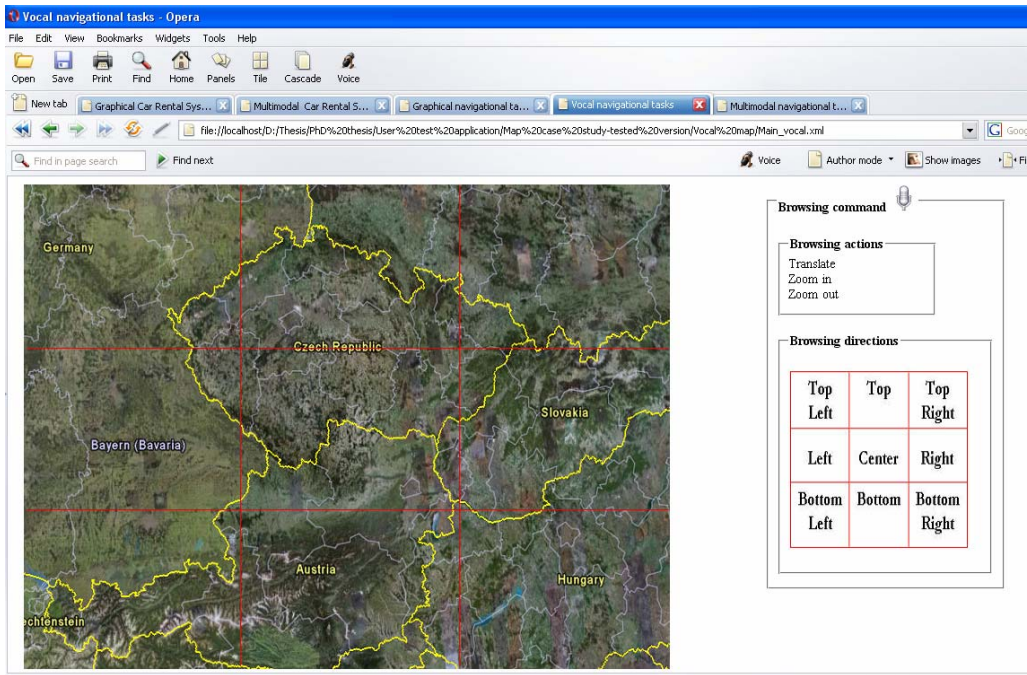


Figure 6-28 FUI – vocal input

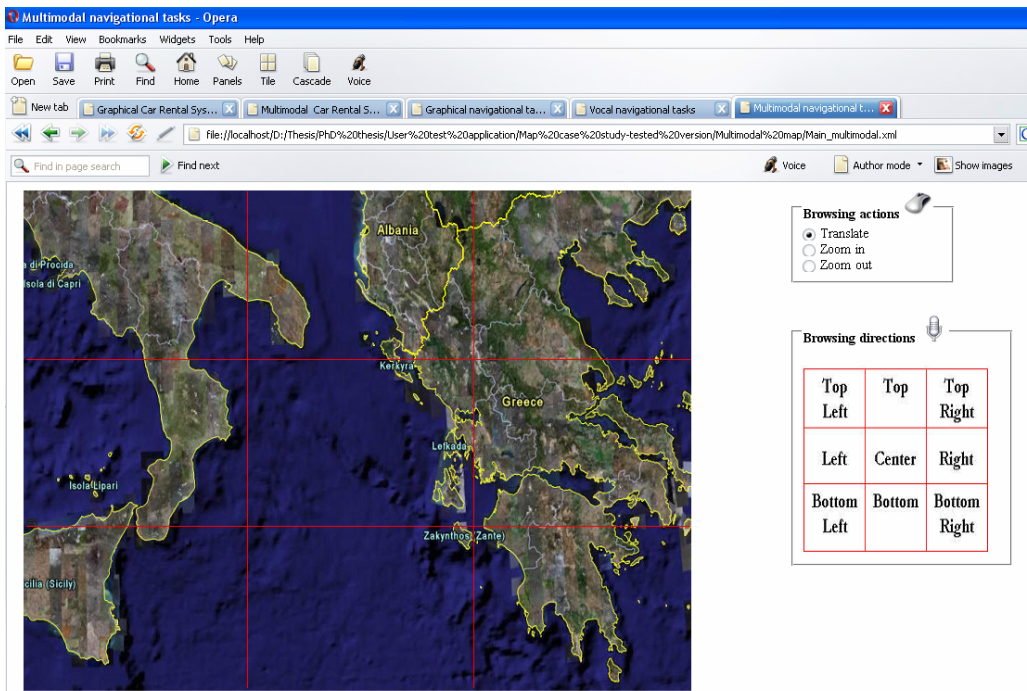


Figure 6-29 FUI – graphical input for browsing action and vocal input for browsing direction

6.5 Empirical validation

6.5.1 Methodology usability assessment

We defined and detailed so far a methodology composed of three dimensions: (1) *the models* based on which a (2) *development method* is applied thanks to the support of (3) *tools*. The previous three sections showed the feasibility of generating MM UIs based on this methodology. This section identifies and discusses the four levels of assessments that can be conducted over the proposed methodology (Figure 6-30).

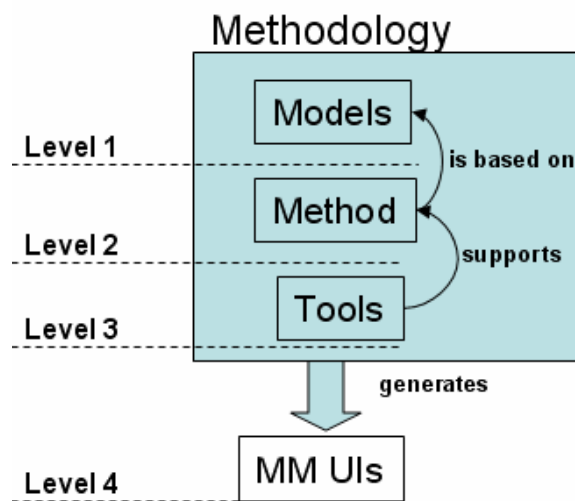


Figure 6-30 Methodology assessment levels

The impact of the development methodology could be assessed by estimating or computing evaluation criteria that may fall into two categories [Olsi04]:

1. *Qualitative criteria*: are typically evaluated in a subjective manner or can be approximated by quantitative metrics. Such criteria include: security, reliability, reusability, usability, etc. For instance, usability could be evaluated in a subjective way through its user subjective satisfaction rate or could be approximated with Usability Evaluation Methods. Usability could be for instance approximated via the IBM CSUQ questionnaire with a correlation of 0.94.
2. *Quantitative criteria* cover criteria that are typically evaluated in an objective manner through metrics. A metric is often referred to as a practical measurement of a product or a process in the software development life cycle of an interactive system. Such metrics could cover project management (e.g., development cost, resources (time, budget, human resources)) and software development. Several methods exist for this purpose. For instance, COConstructive Cost Model (COCOMO) is proposed in two versions (I and II). The first version supports three metrics: source lines of code, function points [Albr83] and use case points.

6. Validation

Ideally, the assesment of an IS development methodology should be conducted for each of the four levels by considering both types of criteria. However, a set of level-specific constraints prevent us to follow this assesment plan. Hereafter, we detail in a structured manner the aims and goals of each level, the proposed assesment plan and their specific constraints.

6.5.1.a Level 1. Model assesment

Aims and goals. Evaluate the quality of the models that will be further managed by the method.

Assesment plan. Assesment of the models applied by the methodology should be carried out with designers of information systems. The plan could involve the evaluation of the support level offered by the proposed models for software requirement specification that respond to the set of features (i.e., unambiguous, complete, verifiable, consistent, modifiable, traceable and usable during the operational phases) identified in [Meye88]. This would help to avoid the seven sins of the software analyst emphasized in [Meye85].

Constraints. There are two important reasons that prevent us conducting such an evaluation. First, there is still a lack of MM UIs and consequently there is a small number of professionals involved in their design. Therefore, they are very solicited persons and difficult to involve in assesment studies. Second, evaluation experiments tipically require extended financial resources especially when the participants are difficult to find.

6.5.1.b Level 2 Method assesment

Aims and goals. Evaluate the understandability (i.e., ease to perceive, ease to apply, lack of confusion generation) and reproductibility (i.e., if two experts are provided with the same case study, similar results should be obtained) of the method.

Assesment plan. Assesment of the method applied by the methodology should be carried out with designers having prerequisites in applying structured development methods (i.e., UML or alike). A training session should be ensured by already experienced proffesionals in order to provide an in-depth understanding. A subjective estimation time for mastering the different methodological aspects is provided in Table 6-35.

Methodological aspects	Estimated learning time
Task Modeling	1/2 day
Domain Modeling	1/4 day
Mapping Modeling	1/4 day
Selecting and using Design Options	1 day
Performing Transformation Rules	1 day
Total	3 days

Table 6-35 Estimated learning time of the methodological aspects

6. Validation

Constraints. In order to be valid the assesment should be carried out with a high number of designers having different levels of expertise. They should apply the methodology with and without the help of the proffesionals, over case studies with different complexity levels (i.e., low, medium and high). The obtained results should be analysed so that to provide a comparision based on a set of metrics. This evaluation process is highly complex and therefore is very hard to achieve. Moreover, the constraints identified in the previous level are perfectly valid in this context as the assesment of both levels should be conducted with the same stakeholders (i.e., the designers).

6.5.1.c Level 3. Tool assesment

Aims and goals. Evaluate the integrity of the tool with respect features such as level of method support, easiness to use the tool, user-friendly related aspects, etc.

Assesment plan. Assesment of the tools should be carried out with a high number of developers having a proven experience with information systems development tools.

Constraints. In order to be valid the assesment should be carried out with a high number of designers by considering criteria such as levels of expertise, organizations, country [Bygs07]. The test should consider case studies with different levels of complexity (i.e, low, medium and high) that generate ISs with or without the tool. Based of a set of metrics (e.g., the function point) a comparision of the resulting ISs should be achieved. Moreover, acceptance test measures [Shne98] (i.e., time for users to learn specific functions, speed of task performance, rate of errors by users, user retention of commands over time, subjective user satisfaction) could be considered.

However, this evaluation proces implies a high complexity and therefore is very hard to accomplish. In addition, such experiment would suppose to provide designers with the assembly of tools presented in Chapter 5 for which a very time-consuming training session should be foreseen. Moreover, due to rather limited financial resources at our disposal we find it difficult to involve a sufficient number of developers that could render our evaluation statistically valid.

6.5.1.d Level 4. Result assesment

Aims and goals. Taking into account the constraints identified in the previous levels, we decided to asses the results produced by our methodology based on an empirical study with end-users. The final goal of this evaluation wasn't to determine the interaction (combination of interaction) modalities that is the most preffered by the end-users, but rather to validate the results produced by the methodology and to provide an idea of the relative usability level generated by different design option values. Therefore, this section describes the participants to the experiment (called from now on the subjects), the set up used to conduct the

6. Validation

study, whereas the next section will detail the usability assessment process and will provide a discussion over the obtained results.

The subjects. The test involved 20 non-native English speakers out of which 10 experienced and 10 inexperienced with respect to the alternative means of interacting with information systems (i.e., speech, tactile screens, joysticks) other than mouse and keyboard (Table 6-36). The gender was equally assigned so that 5 males and 5 females participated for each category. The average age of the subjects is 29.

N° subjects	Average age	Gender		Multimodal interaction experience	
		Male	Female	Experienced	Inexperienced
20	29	10	10	10	10

Table 6-36 Summary of the subject's demographics and experience level

The tested applications. The subjects were asked to test two types of applications designed in the English language. In order to avoid the fatigue of the subjects the experiment took place in two sessions organized in two different days as follows:

- **Session 1 - Web form applications:** two applications were tested according to a predefined scenario:
 - *Car rental:* the case study described in Section 6.3.
 - *DVD rental:* we do not describe the development life cycle of this application as it has the same complexity level as the car rental system and considered the same interactions. It enables users to rent a DVD (Figure 6-31). For this purpose, a set of data such as rental date and movie type had to be filled in based on which the application provides the available movies. Once a movie is selected, the subjects had to specify the payment information (e.g., owner's name, credit card type, number and expiration date).

6. Validation

The screenshot shows a web browser window with a blue header bar. Below the header, there is a navigation bar with a microphone icon and the text "Voice". The main content area is titled "DVD rental preferences" and contains two panels. The left panel, "Rental date", has three dropdown menus: "Day" (set to 1), "Month" (set to January), and "Year" (set to 2007). The right panel, "Movie type", has four radio buttons: "action" (selected), "comedy", "drama", and "thriller". Below these panels is a "Search available movies" button. The next section is "Movie selection", which shows a list of movies: "The lord of the rings" (highlighted), "Lost", "Home alone", and "In bed with the enemy". The final section is "Payment information", which includes a "Name" text input, a "Card type" dropdown menu (set to VISA), a "Card number" text input, and an "Expiration date" section with "Month" (January) and "Year" (2007) dropdown menus. At the bottom of the form are "OK" and "CANCEL" buttons.

Figure 6-31 The multimodal version of the DVD rental application

For both applications the task was considered achieved when the participants confirmed the payment by clicking on the OK button. The subjects had to test three types of interaction in a random order.

- **Session 2 - Map browsing:** is the application described in Section 6.4. Previously to the experiment, the subjects were submitted to a test concerning their geographical knowledge about the position of the main European capitals. The accepted subjects were asked to achieve two different tasks (i.e., searching an European capital on the map) per interaction modality in a random order:

6. Validation

The task was considered achieved when the participants were able to point out on the map the name of the capital specified by the scenario.

Apparatus and experimental environment. The physical position of the subjects and apparatus involved in the test are sketched in Figure 6-32. The notebook used in this study was a PC Dell Latitude D820 equipped with an Intel Core 2 Duo T7200 (2.0 GHz, 4 Mo cache level 2 memory) processor and 2 GB of RAM memory. The 15" screen was set to 1280 by 800 pixels resolution, with a 32-bit color palette. The integrated loud speakers were used for voice output, while Philips SBC 90 microphone fasten on a tripod enabled voice input thanks to the audion input port of the notebook. The Microsoft Wireless Notebook Optical Mouse 4000 was connected to the USB port. A Sony video camera oriented towards the notebook's screen was used for video recording.

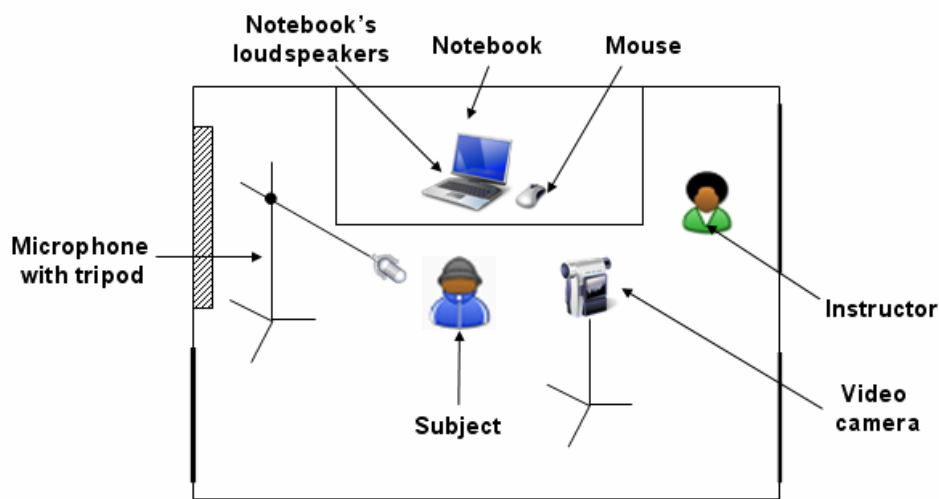


Figure 6-32 Physical position of the subjects and experimental apparatus

The tested applications were specified in XHTML v1.0 (for the graphical interaction), whereas X+V v1.2 language was employed for vocal and MM interactions. All the applications were interpreted with the Opera browser v9.24 embedded with IBM Multimodal Runtime Environment v4.1.3 using ViaVoice speech technology.

The instructor welcomed the subjects (Figure 6-33) and provided the instructions for the evaluation test. He also supervised the good unfolding of the experiment (e.g., clarifies the upcoming issues, takes notes during the test, unblocks the subjects if they were unable to complete their task, checks the responses given by the subjects).



Figure 6-33 A subject interacting with the application

6.5.2 Methodology result assesment plan

The plan is split in three phases describing the procedure followed before, during and after the test.

6.5.2.a Pre-test

Before starting the test, the participants were guided by the instructor with the testing procedure as follows:

1. Subjects were given information about the goals of the test. They were informed that it is not a test of their abilities and that their interaction with the applications will be video recorded in order to be further analysed. Finally, they were announced that a tracking number will be assigned to each of them for identification purposes during the entire procedure.
2. Subjects were asked to fill in a demographic questionnaire that gathered information related to their gender, age, activity field, years of experience. They also scored their attitude towards technology in general by specifying the frequency of using a computer and their experience with MM interaction.
3. Subjects were further advised to try accomplishing the tasks without any assistance. However, they were allowed to ask for help only if they felt unable to complete the task on their own.
4. The instructor responded to the subject's questions, if any and checked whether they understood well the instructions or not.
5. Finally, the subjects were asked to complete the tasks according to the given instructions. To avoid a possible ceiling effect, there was no time limit to complete the tasks.

6.5.2.b Test

The test was split in two steps:

- The first step consisted of a training session assisted by the instructor who explained how to carry out the task. This enabled subjects to discover the MM capabilities of the Opera browser and the correct manipulation of the voice-enabling button. The training application emerged from our methodology and its UI contained most of the interaction objects (Figure 6-34) that were further present in the tested application. The training session had no time limit.

The screenshot shows a web browser window with a blue header bar. Below the header, there is a navigation bar with a 'Voice' button and a 'Training System' button. The address bar shows the URL: 'ID%20thesis/User%20test%20application/Training%20system/training_multimodal.xml'. The main content area contains a form titled 'Personal information' with the following fields:

- First name**: Text input field with a microphone icon.
- Last name**: Text input field with a microphone icon.
- Gender**: Radio buttons for 'male' and 'female', with a microphone icon.
- Birth day**: A form with three dropdown menus for 'Day' (set to 1), 'Month' (set to January), and 'Year' (set to 1960), with a microphone icon.
- Hobbies**: A list of checkboxes for 'travels', 'music', 'sport', 'movies', 'reading', and 'games', with a microphone icon.

Below the 'Personal information' form is a section titled 'Favourite european capital' with a list box containing 'Paris', 'Brussels', 'Madrid', and 'Rome', and a microphone icon. At the bottom of the form, there are 'OK' and 'CANCEL' buttons with a mouse cursor icon.

Figure 6-34 The training application

- In the second step, the subjects were asked to proceed with the test according to a random selection of the scenarios. The task was expected to be accomplished without any assistance. However, if the subjects got stuck with one of the tasks and felt unable to continue, they could ask for help. The instructions given to the users clearly stated that they should only ask for assistance as a last resort. When the subject asked for help, the instructor explained the next step that the user needed to take.

6. Validation

6.5.2.c Post-test

At the end of the test, the subjects were interviewed according to a structured scheme. First, they were asked to rank their preference on a scale from 1 (minimum) to 7 (maximum) for each type of interaction modality. Further, during a debriefing session, the subjects were asked to specify three positive and three negative aspects with respect to the employed interaction modalities. For this purpose they were able to reuse the applications in order to better explain the encountered feelings and difficulties during the experiment. At the end of each session, subjects were given a small reward for their participation.

6.5.3 Results

6.5.3.a Evaluation measures

So far, the reasearch community did not manage to create an authoritative list of MM usability parameters. Moreover, we are not aware of any hierarchy of concepts of usability properties whose parameters and values are known. Therefore, the final purpose of our experiment was to measure the relative usability level among the tested interaction modalities thanks to a subset of usability parameters considered from an empirically collected parameter list [Bern06]. The subset is composed of six parameters that were considered, entirely or partially, for the analysis of the interaction modalities employed in the two types of applications:

1. *Task completion time*: measures the interaction efficiency.
2. *Task percentage completion*: measures the effectiveness of the application with respect to the employed interaction modality.
3. *Error rate*: measures the interaction efficiency in terms of number of errors per interaction modality.
4. *Learning time*: measures the ease of learn of a particular interaction modality.
5. *Number of mouse clicks*: measures the interaction efficiency in terms of number of clicks per interaction modality.
6. *Interaction modality preference*: measures the relative satisfaction among the tested interaction modalities.

6.5.3.b Web form applications

6.5.3.b.1 Task completion time

The task completion time was computed as a mean (M) time between the two applications of type form for each interaction modality. Figure 6-35 shows that the mean time to achieve the tasks using the graphical modality (M= 74.58 seconds) is comparable with the the MM interaction (M= 94.63 seconds). An important observation is that, for the latter interaction half of the subjects chose a real MM interaction (i.e., the vocal is combined with the graphical modality), while

6. Validation

the other half achieved the task using just the graphical modality. The vocal modality required a considerable longer time ($M = 211.88$ seconds), which represents 228% more than the graphical interaction and 174.2% more than the MM interaction.

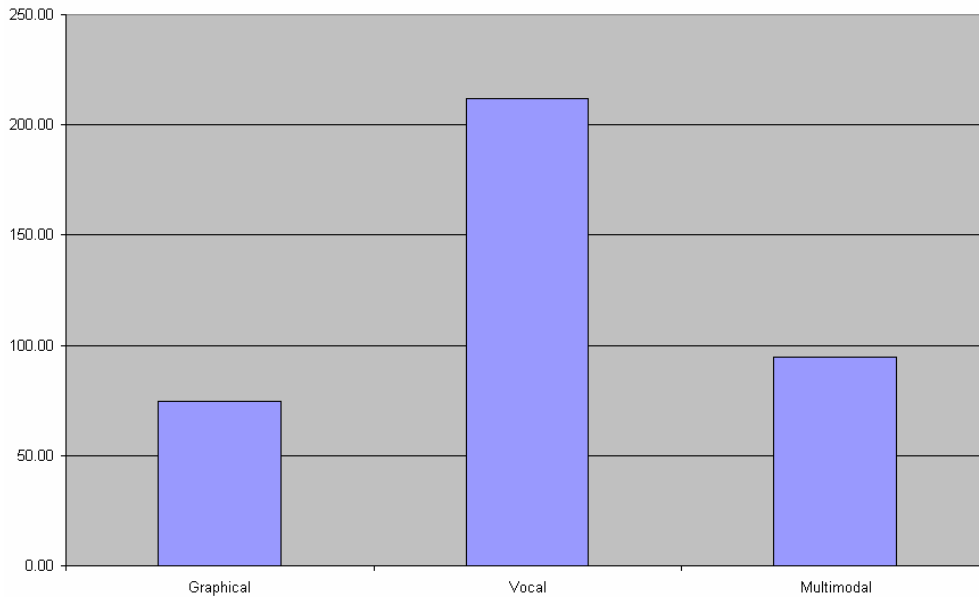


Figure 6-35 Task completion mean time per interaction modality

6.5.3.b.2 Task percentage completion

The task percentage completion was computed as a mean between the two applications of type form for each interaction modality (Figure 6-36). While the completion rate using the graphical and the MM interactions are approximately equal ($M = 99.56\%$ and $M = 99.68\%$, respectively), the vocal interaction rate is slightly lower ($M = 96.4\%$).

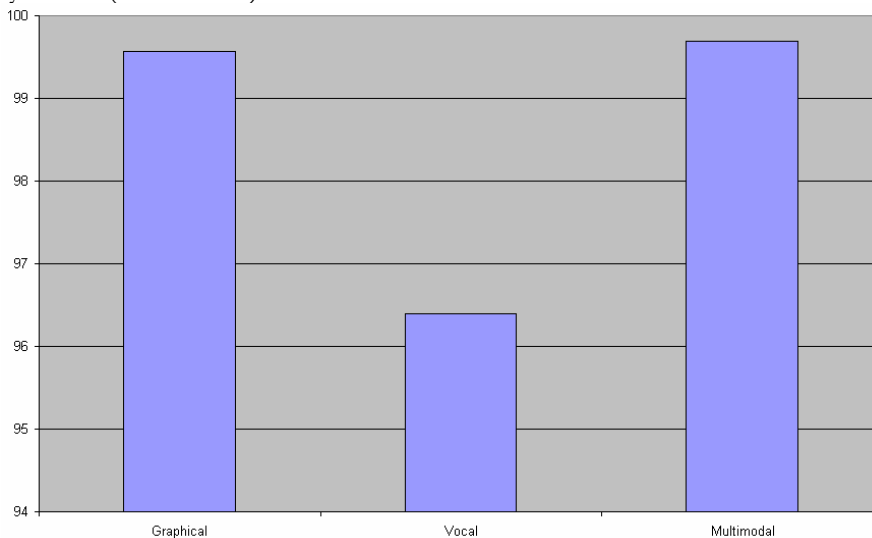


Figure 6-36 Mean task percentage completion per interaction modality

6. Validation

6.5.3.b.3 Error rate

During the usability study, we identified 7 categories of errors that were computed as a mean between the errors of the two web applications for each interaction modality (Figure 6-37). All the categories are meaningful for vocal and MM interaction, while for the graphical interaction only the errors that fall in the 5th and 6th categories were considered. However, the latter interaction no errors were observed and, consequently, we don't illustrate them in the chart. For the vocal and MM interactions the results are as follows:

1. *Synchronization*: errors due to the order violation of the interaction states to follow when employing the voice-enabling button in Opera browser (M = 2.3 errors and M = 0.65 errors, respectively). A correct sequence of the states to reach by the button is illustrated in Figure 6-38 on a timeline: the button in the initial state is further pushed in the 2nd state so as to enable the voice recognition engine in state 3. User's utterance is afterwards possible. Once that the utterance is ended the button has to be released in state 5. The voice recognition is disabled after a short period (state 6). Afterwards, the initial state is reached again.
2. *Pronunciation*: errors due to a bad pronunciation of the commands (M = 2.25 errors and M = 0.4 errors, respectively).
3. *No input*: errors due to the lack of vocal input once the voice-enabling button reached the 3rd state (M = 0.42 errors and M = 0.1 errors, respectively).
4. *No match*: errors due to vocal inputs that were correctly pronounced but were not predicted in the associated grammar as they are not context-meaningful (M = 0.07 errors and M = 0.05 errors, respectively).
5. *Irrelevant actions*: errors due to an incorrect manipulation of the Opera browser (M = 0.8 errors and M = 0 errors, respectively). For instance, clicking on the zoom level interaction field before uttering a number in the application, which determined the modification of the application layout.
6. *System*: errors due to a bad design of the UIs (M = 0.65 errors and M = 0 errors, respectively). For instance, an incorrect pronunciation of the available options to select with vocal interaction, which induced the user to errors (e.g., the pronunciation of the year 2012 as "twenty twelve").
7. *Noisy environment*: errors due to undesirable sounds produced in the vicinity of test room while the voice-enabling button was in state 3 (M = 0.02 errors and M = 0, respectively).

6. Validation

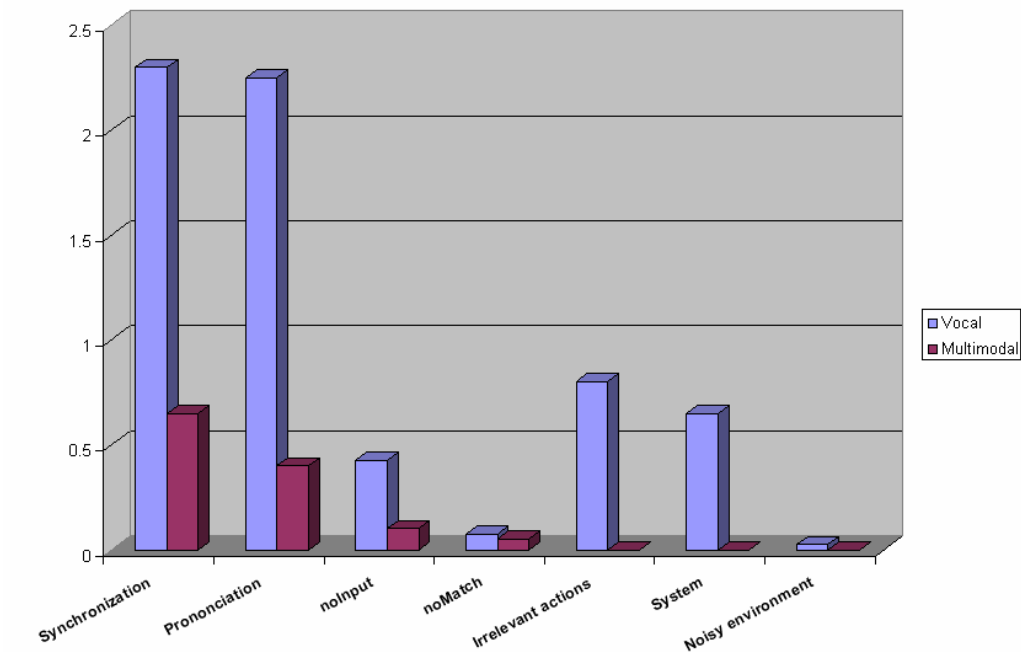


Figure 6-37 Mean number of errors per category

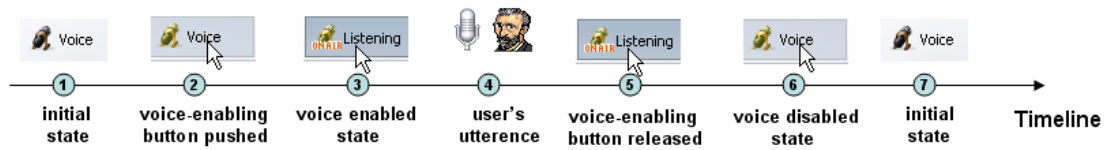


Figure 6-38 The timeline for a correct voice-enabling button manipulation

6.5.3.b.4 Learning time

During the first test, we measured the task completion time for vocal interaction in the second application of type form. We asked the subjects to repeat this test following the same scenario after a couple of days. The results illustrated in Figure 6-39 show that 75% of the subjects improved their time, 10% had the same performance, while rest had a longer time. Consequently, the task completion mean time for the second test ($M = 95.50$ seconds) is significantly lower than the first test ($M = 131$ seconds).

6. Validation

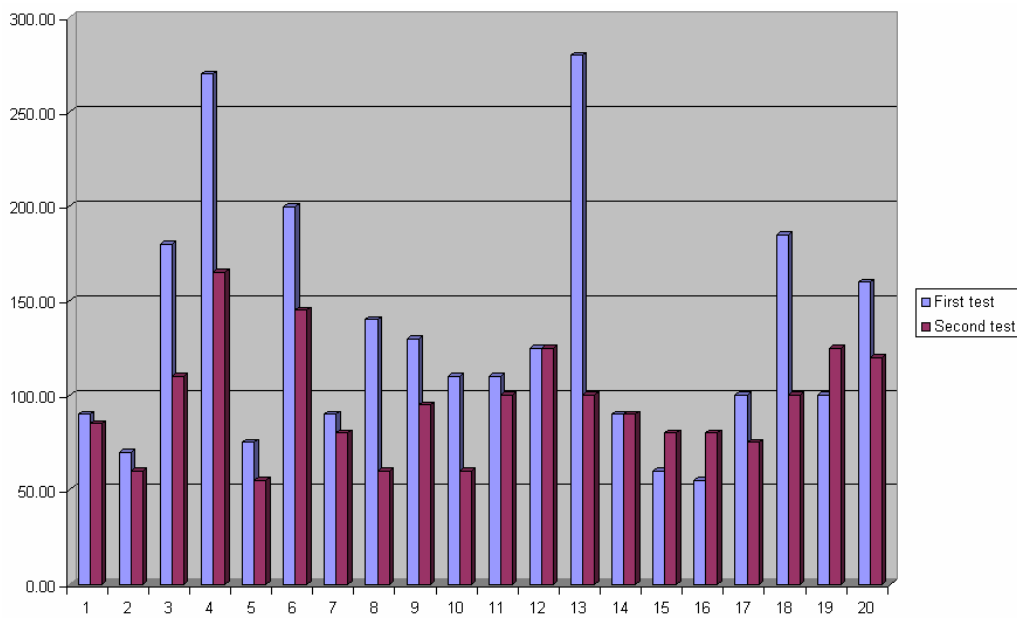


Figure 6-39 Learning time for vocal interaction

6.5.3.b.5 Number of mouse clicks

The mean number of mouse clicks was computed as a mean between the mouse clicks of the two web form applications for each interaction modality. Figure 6-40 shows that the mean for graphical and MM interactions are approximately equal ($M= 27$ clicks and $M= 26$ clicks, respectively), whereas the vocal interaction requires a sensibly greater number of clicks ($M= 32$ clicks).

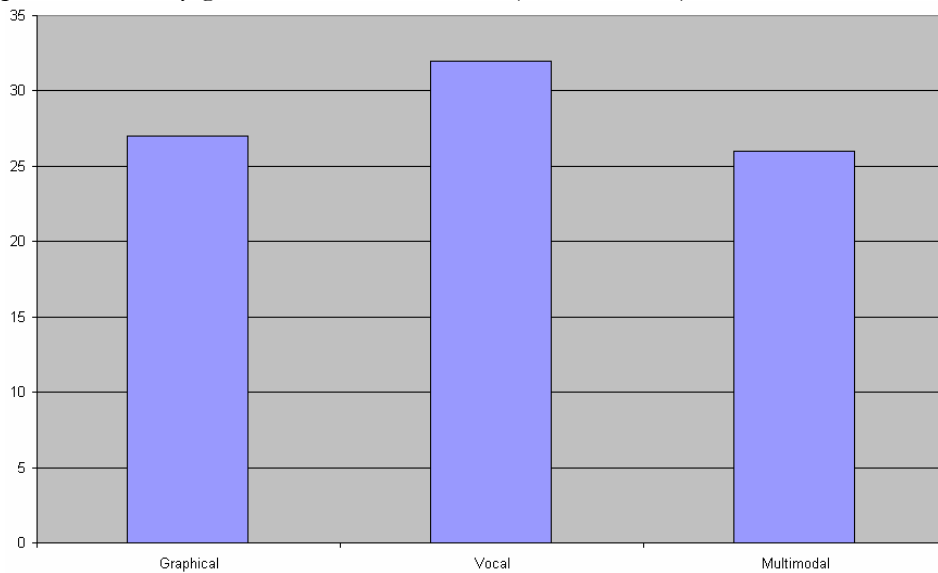


Figure 6-40 Mean number of mouse clicks per interaction modality

6. Validation

6.5.3.b.6 Interaction modality preference

The distribution of the modality preference per subjects illustrated in Figure 6-41 shows that the graphical (M= 6.5) and the MM (M= 6.2) interactions were approximately equally ranked, whereas the vocal interaction was the less preferred (M= 3.6).

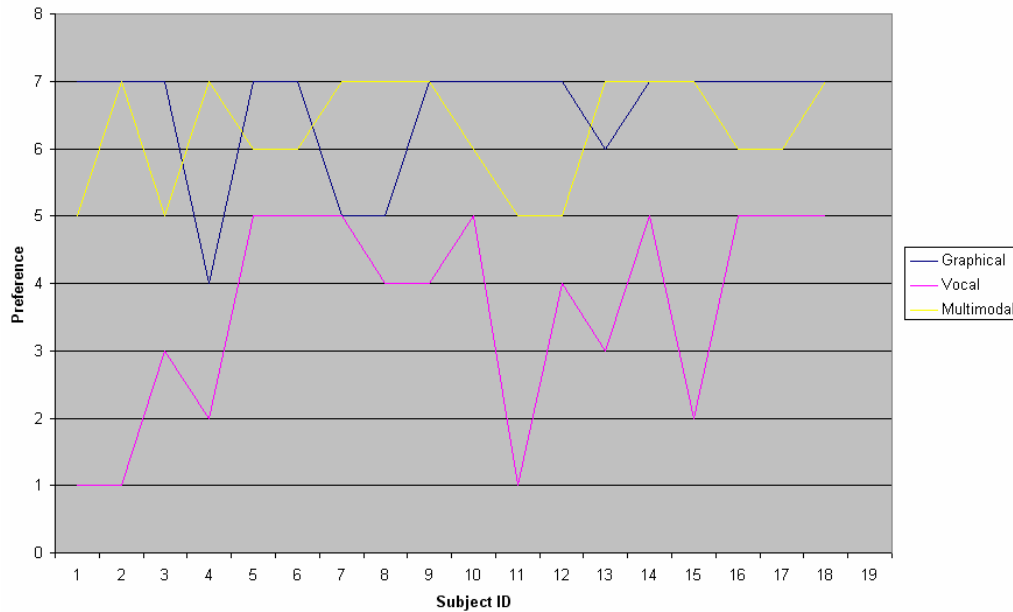


Figure 6-41 Distribution of the modality preference per subject

6.5.3.c Map browsing application

The current section presents the statistical outcome of the map browsing application. Due to the fact that the tasks for the three types of interaction were different in order to avoid the learning effect, the comparison between the modalities doesn't make sense. Instead, the differences between the experience groups with respect to the following usability parameters considered for the same interaction modality are analysed.

6.5.3.c.1 Task completion time

The task completion time was computed for each type of interaction as a mean between the completion time of the two considered tasks. While for the graphical interaction the mean time between the two groups is not significant, for the vocal and MM interaction the experienced subjects proved to be faster (Table 6-37).

6. Validation

Interaction modality \ Experience group	Graphical	Vocal	Multimodal
Experienced	73	83.5	156.75
Unexperienced	74.75	98	192

Table 6-37 Mean task completion time (seconds) per experience group

6.5.3.c.2 Task procentage completion

The task completion procentage was measured for each interaction modality as a mean completion procentage of the two considered tasks. While for the graphical interaction there is a 100% succesful rate for both experience groups, for vocal and MM interaction the experienced subjects proved to have a better completion rate (Table 6-38).

Interaction modality \ Experience group	Graphical	Vocal	Multimodal
Experienced	100 %	97 %	156.75 %
Unexperienced	100 %	85.75 %	192 %

Table 6-38 Mean task procentage completion per experience group

6.5.3.c.3 Number of errors

For the second session we considered the same categories of errors and their assigned meaning as those described in Section 6.5.5.b.3. The number of errors was computed as a mean between the errors of the two tasks considered for each interaction modality. While for the graphical interaction no errors were observed, for vocal (Figure 6-42) and MM interactions (Figure 6-43) the results show a relatively higher number of synchronization errors compared to the other error categories for both experience groups.

6. Validation

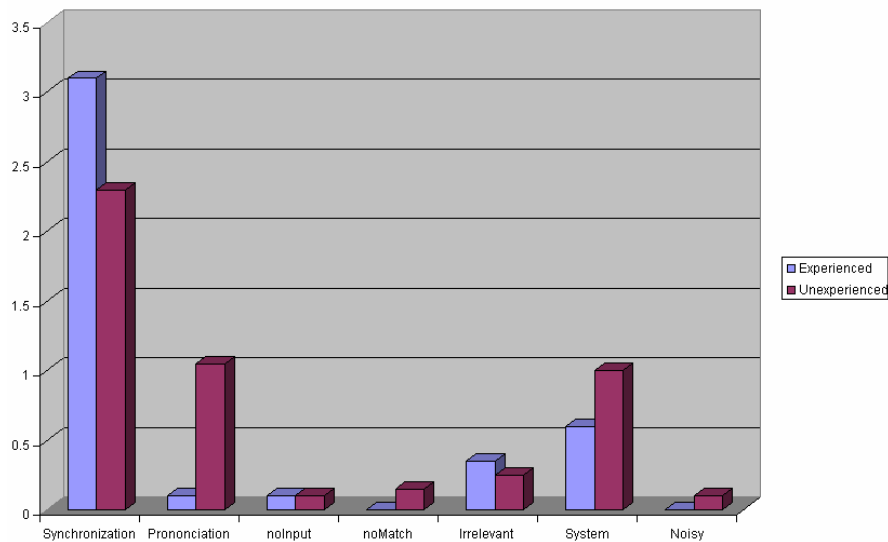


Figure 6-42 Mean number of errors per experience group for vocal interaction

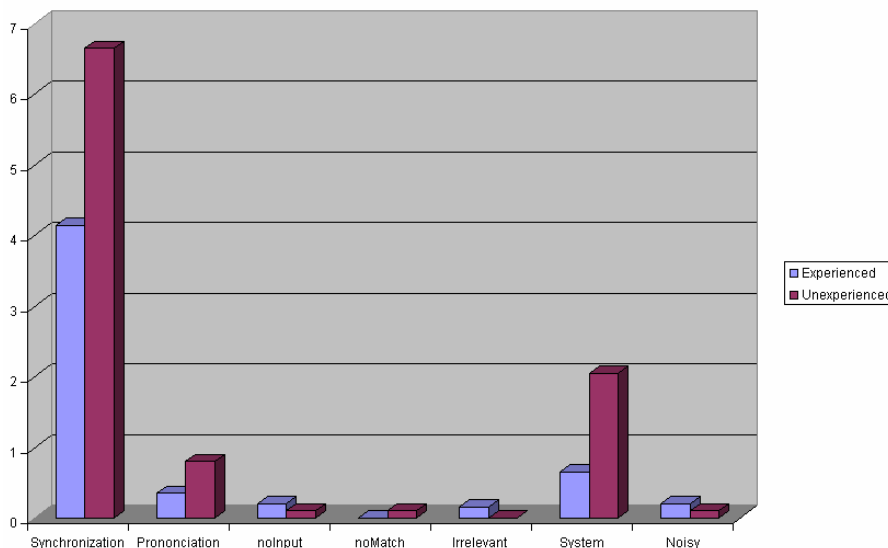


Figure 6-43 Mean number of errors per experience group for MM interaction

6.5.4 Interpretation and discussion

6.5.4.a Web form applications

In order to assess and compare the distribution of the time to achieve the task for the two groups, a box plot graph is illustrated in Figure 6-44. We notice that the dispersion of time values for MM and vocal interaction is higher than the graphical one for both groups. The experienced subjects employing the MM interaction had a dispersion ranging from 65 seconds to 107.5 seconds with more values situated above the median time of 75 seconds. For vocal interaction, the

6. Validation

same group had a higher dispersion within an interval from 131.25 to 243.75 seconds, but still with more values above the median time of 176.25 seconds.

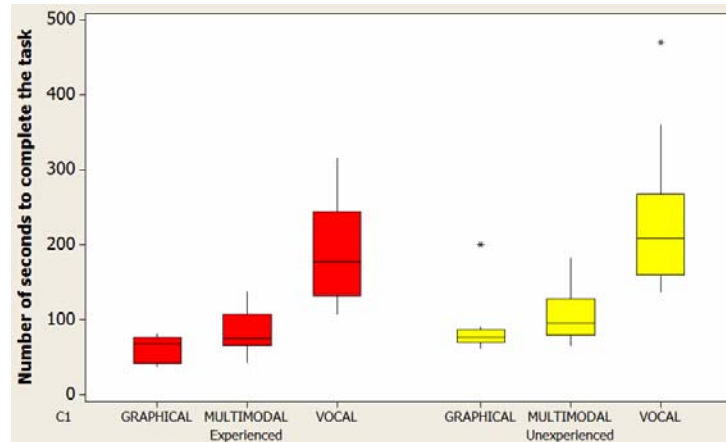


Figure 6-44 Distribution of task completion time per interaction type and experience group

In order to test if MM interaction experience had a significant influence over the mean task completion time a t-Test analysis was conducted (Table 6-39). The results show that this was the case only for the graphical interaction ($p=0.0339$). However, the experimented subjects proved to be faster than the non-experienced ones (Figure 6-45) for the three types of interaction ($M=61.25$ seconds vs. $M=87.90$ seconds for graphical, $M=188.0$ seconds vs. $M=235.75$ seconds for vocal interaction and $M=84.0$ seconds vs. 105.25 seconds for MM interaction).

t-Test	Significant difference with respect to MM experience
Graphical	$p=0.0339 < 0.05$
Vocal	$p=0.1207 > 0.05$
Multimodal	$p=0.0901 > 0.05$

Table 6-39 t-Test results for the significant difference in mean task completion time with respect to MM experience

6. Validation

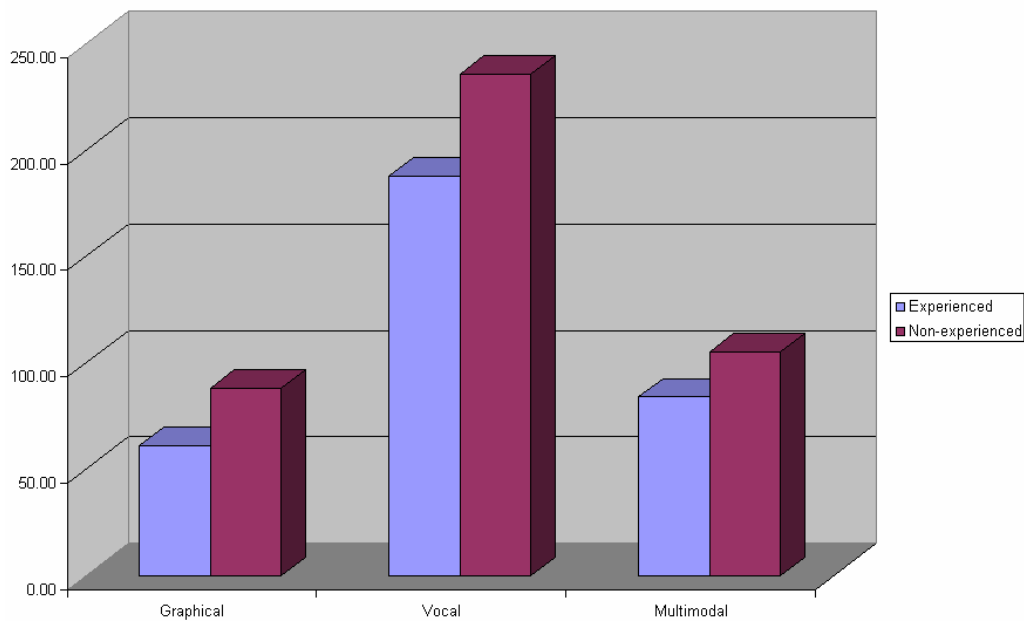


Figure 6-45 Task completion mean time per experience group

The statistical analysis revealed a slower task completion time for both vocal and MM interactions. This was mainly due to synchronization and pronunciation errors, resulting from the difficulties of manipulating the voice-enabling button and from the fact that subjects were non native English speakers. The Pearson correlation coefficient [Sieg88] between the average of the pronunciation and synchronization errors and the mean completion time sustains the above statement for both vocal (Pearson= 0.75) and true MM interactions (Pearson= 0.73). In addition, the same categories of errors are influencing the number of mouse clicks (Pearson= 0.77 for vocal interaction and Pearson= 0.63 for true MM interaction) which had a negative impact over the efficiency of the interactions. However, the t-Test analysis (Table 6-40) revealed that there is no significant difference in the mean number of pronunciation and synchronization errors between the two groups of subjects neither for vocal, nor for the MM interactions.

t-Test	Significant difference with respect to MM experience
Vocal	$p=0.1603 > 0.05$
Multimodal	$p=0.2459 > 0.05$

Table 6-40 t-Test results for the significant difference in mean number of pronunciation and synchronization errors with respect to MM experience

6. Validation

6.5.4.b Map browsing application

As the results in Figure 6-45 show that the experienced subjects were faster than the unexperienced ones, we examined whether the experience category has a significant influence over the task completion time thanks to a t-Test analysis. The results presented in Table 6-41 show no influence whatsoever.

t-Test	Significant difference with respect to experience category
Graphical	$p = 0.1074 > 0.05$
Vocal	$p = 0.2321 > 0.05$
Multimodal	$p = 0.1675 > 0.05$

Table 6-41 t-Test results for the significant difference in mean completion time with respect to experience category

In line with the observations made in Section 6.5.6.a, the statistical analysis revealed a high number of synchronization errors. The Pearson correlation function between the average number of synchronization errors and the mean completion time sustains the above statement for both vocal (Pearson= 0.60) and MM interactions (Pearson= 0.58). In addition, the average between the synchronization and pronunciation errors influence the task percentage completion (Pearson= -0.72 for vocal interaction and Pearson= -0.48 for MM interaction) which leads to a negative impact over the efficiency of the interactions. However, the t-Test analysis revealed that there is no significant difference in the mean number of synchronization errors between the two experience categories neither for the vocal interaction, nor for the MM one.

6.5.4.c Overall interpretation

Thanks to the debriefing session, we were able to illustrate the modality preference for each type of application (Figure 6-46). The graphical interaction was the most preferred interaction for both application types. While for the web form applications the MM interaction is better ranked, the vocal interaction takes the lead when it is employed for the map browsing application. This is particularly due to the structure of the navigational commands employed in the browsing application. While for the vocal interaction both the action and the parameters are specified as a whole in one utterance, the complementarity nature of the MM command where the action is specified graphically and the parameters vocally requires a modality break that slows down the subjects.

6. Validation

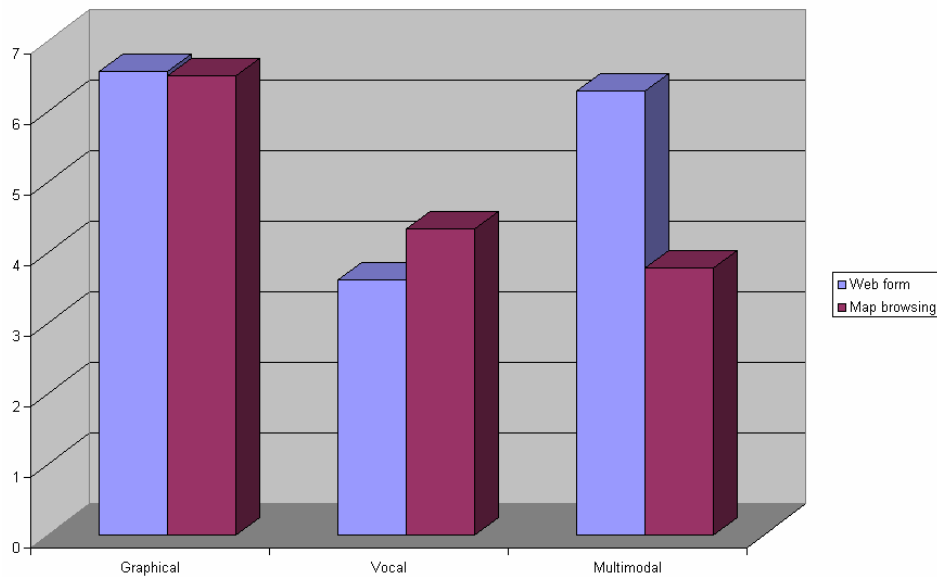


Figure 6-46 Modality interaction preference per application type

The subjects were also asked to specify three positive and three negative aspects with respect to the tested interaction modalities. The results in Table 6-42 are specified along with their frequency of appearance. 75 % of the subjects preferred the graphical interaction as it is a day to day habit, thus enabling a faster and more intuitive interaction.

Even if the vocal interaction was less preferred, more than 75% of the subjects appreciated it especially for widgets that required multiple selections at once (e.g., the group of three comboboxes that enable to specify the date, groups of checkboxes) or one single selection that requires a long scroll (e.g., a combobox or a listbox where the item to select is at the end of a long list). Moreover, 60 % of the subjects felt attracted by the interaction as it is more natural to use and closer to the human-to-human interaction. It also urges the user to experiment as it is seen as an escape from the habitual interaction modalities.

The MM interaction was particularly appreciated by 80% of the subjects for its flexibility in choosing the appropriate interaction (i.e., either graphical or vocal) depending on the widget, which enabled to take advantage of the positive aspects of each interaction type.

In terms of negative aspects, the subjects didn't have any remarks for the graphical interaction. Instead, 50% of them complained about the high number of clicks when employing the vocal interaction which has a negative impact over the efficiency of the task achievement. They equally pointed out difficulties of synchronization with the voice-enabling button which affected the effectiveness of the interaction. Most of them would have like to have a continuously active recognition engine that eliminates the constraint of pressing the button. They also confessed that the high pronunciation error rate is probably due to the lack of

6. Validation

English pronunciation skills. For the MM interaction the same negative aspects detailed above were emphasized when the selected interaction was the vocal one.

Interaction type Aspects	Graphical interaction	Vocal interaction	Multimodal interaction
Positive	<ul style="list-style-type: none"> • Habit (15) • Fast (10) • Intuitive (3) • Simple (2) 	<ul style="list-style-type: none"> • Fast for multiple selections and for single selection in a long list of items (17) • More natural and human (12) • Urges to experiment it (4) • Out of habit (2) • Fun (1) • Very didactic (1) 	<ul style="list-style-type: none"> • Flexibility to select the appropriate modality (16)
Negative	-	<ul style="list-style-type: none"> • High number of clicks to specify a vocal input (10) • Difficult to synchronize with the the voice-enabling button (10) • Pronunciation errors for non native english speakers (4) • Lack of habit (2) 	<ul style="list-style-type: none"> • When the vocal interaction is employed its negative aspects are inherited

Table 6-42 Subject's opinion for web form applications

At the end of the second session, the subjects were again asked to provide three positive and three negative aspects with respect to the tested interactions (Table 6-43). 60% of them considered the graphical interaction a habit that makes the interaction faster and intuitive.

The vocal interaction was characterized by 75% of them as being fast and comfortable thanks to the all at once commands that are easy to utter. While the graphical modality requires to focus on the browsing elements to select, the vocal interaction escapes from this shortcoming as it enables to focus on the map details and to think about the next command while uttiring the current one. The MM interaction was particularly appreciated for better recognition probably due to the shorter commands to utter.

The most important negative aspect claimed by 75% of the subjects is the high number of clicks required by the graphical interaction which affects the effectiveness of the application. For the vocal modality most of the negative aspects were consistent with those emphasized for the web form applications. Overall the subjects dislike the MM interaction due to the modality break encountered while specifying the command's components. Therefore, 75% of the subjects confessed that they would have preferred either the graphical interaction or the vocal interaction with a lower error rate.

6. Validation

Interaction type Aspects	Graphical interaction	Vocal interaction	Multimodal interaction
Positive	<ul style="list-style-type: none"> • Habit (12) • Fast (6) • Intuitive (4) 	<ul style="list-style-type: none"> • Fast and comfortable thanks to all at once commands (15) • Easy to pronounce commands (10) • Enables to focus on the map without paying attention to the details concerning the command's build up (8) • Enables to think about the next command while uttering the current one (6) • Better guided than the graphical interaction (5) • Fun (5) 	<ul style="list-style-type: none"> • Better recognition thanks to shorter commands to utter (8) • Enables to think about the direction to browse while making the graphical selection (6) • Better guided than the graphical interaction (5)
Negative	<ul style="list-style-type: none"> • High number of clicks to specify the commands (15) • Need for drag and drop support (2) 	<ul style="list-style-type: none"> • Difficult to synchronize with the the voice-enabling button while relatively long phrases had to be uttered (10) • Pronunciation errors for non native english speakers (4) • Frustrating (5) • Dislike presing the voice-enabling button (5) • No guidance for the zoom level and position on the map (2) 	<ul style="list-style-type: none"> • Modality break when going from the graphical selection of the browsing action to vocal specification of the direction (15)

Table 6-43 Subject's opinion for non web form application

6.6 Internal validation

The internal validation of a methodology consists of assessing its characteristics against a set of selected criteria. The relevant criteria, called requirements, for our methodology have been elicited and motivated after the state of the art of Chapter 2. This section proposes a discussion for each of these requirements included in the corresponding dimension of the methodology:

Modeling requirements:

Requirement 1. Support for multimodal input/output: states that our ontology should enable multiple (i.e., at least two different) input/output interaction modalities. The current requirement is motivated by the definition of the multimodal systems (Section 1.3.4).

Discussion: This requirement is achieved thanks to the expansion brought to the existing vocal ontology described in Section 3.4.2.b which provides a larger set of vocal CIOs that cover the requirements of vocal and MM UIs. Moreover, Section 3.4.4.b introduces the synchronization relationship between the graphical and the vocal concepts.

Requirement 2. Separation of modalities: states that the concepts and the specifications corresponding to each modality should be syntactically separated one from the other. The current requirement is motivated by two aspects: (1) flexibility in the development process given by the possibility to specify separately the UI corresponding to each involved interaction modality and to further combine them altogether, (2) reusability, totally or partially, of the specification corresponding to an interaction modality in other applications that employ it. This requirement contributes to the principle of separation of concerns [Dijk76].

Discussion: this requirement is achieved thanks to the semantic separation of graphical CIOs (Section 3.4.1) and vocal CIOs (Section 3.4.2) composing our ontology. Moreover the UsiXML syntax (Section 3.5) ensures a separate specification of graphical and vocal elements describing a MM UI.

Requirement 3. Support for CARE properties concerning the input/output modalities: states that our ontology should ensure the support of the CARE properties for input/output modalities. This requirement is motivated by the design facilities offered by the CARE properties when defining the relationships that can occur between input/output modalities.

Discussion: this requirement is partially achieved:

- *Redundancy in input* and *Complementarity in input/output*: not supported due to the following reasons: (1) they require fusion/fission of modalities which are not addressed by the current thesis (Section 1.4.3), (2) the target language is X+V which doesn't offer support for fusion/fission aspects.
- *Assignment*: supported by either graphical CIOs or vocal CIOs depending on the selected interaction modality supported by design decisions.
- *Equivalence*: supported by both graphical CIOs and vocal CIOs between which a synchronization relationship is specified.
- *Redundancy in output*: supported by both graphical CIOs and vocal CIOs which are employed to provide a redundant output to the user.

6. Validation

Requirement 4. Ability to model a user interface independent of any modality: states that the provided ontology should ensure a level in the development life cycle that allows to specify a modality-independent UI. This requirement is motivated by the increasing number of novel devices and consequently of interaction modalities that will determine the development of new UIs with new modality capabilities. A modality-independent level will also enable to avoid the redeployment of UIs from scratch. This requirement contributes to the principle of separation of concerns [Dijk76].

Discussion: this requirement is fully achieved thanks to the existence of the AUI Model that gathers modality-independent concepts (Section 3.3.3).

Requirement 5. Extendibility to new modalities: states that the ontology should allow the extension with new types of interaction modalities. This requirement is motivated by the constant emergence of new computing platforms, each of them supporting a new set of interaction modalities. This requirement is a principle that we would like to cover, but we are well aware that very complex interactions cannot be supported.

Discussion: this requirement is ensured by the modularity of our ontology where each model describing a particular aspect of the UI is defined independently of the other and by the separation of concepts assigned to different modalities.

Requirement 6. Ontology homogeneity: states that the ontological concepts should be defined according to a common syntax. The requirement is motivated by the necessity of defining a single formalism for model concepts in order to facilitate their integration and processing.

Discussion: this requirement is achieved thanks to the selection of UsiXML language (Section 3.2.2) as a unique formalism to support the ontological concepts considered in this thesis.

Requirement 7. Human readability: states that the proposed ontology should be legible by human agents. The current requirement is motivated by two aspects: (1) the need to define in an explicit manner the ontological concepts in order to ensure their precise comprehension, (2) the necessity of sharing the underlying concepts among the research community.

Discussion: this requirement is achieved by employing UML class diagrams in order to specify the semantics of our ontology. Thus, the composing concepts are expressed through classes and relationships between them by providing detailed definitions of their attributes. It is out of the scope of this dissertation to discuss the expressivity of UML notations.

6. Validation

Method requirements:

Requirement 8. Approach based on design space: states that our development life cycle towards a final multimodal UI should be guided by a set of design options. This requirement is motivated by the need to clarify the development process in a structured way in terms of options, thus requiring less design workload.

Discussion: this requirement is achieved thanks to the definition of a design space composed of design options (Section 4.2) that support the designer's decisions during the development life cycle.

Requirement 9. Method explicitness: states that the component steps of our methodology should define in a comprehensive way their logic and application. This requirement is motivated by the lack of explicitness of the existing approaches in describing the proposed transformational process.

Discussion: this requirement is ensured by several factors: (1) human readability of the ontological is a pre-requisite of methodological explicitness, (2) decomposition of the transformational approach into development steps and sub-steps (Section 4.4), (3) existence of a well-defined syntax for expressing methodological steps (Section 3.5).

Requirement 10. Method extendibility: refers to the ability left to the designers to extend the development steps proposed in a methodology. The current requirement is motivated by the lack of flexibility in the current methodological steps that hinders designers to add, delete, modify and reuse these steps.

Discussion: Transformation systems and transformation sub-steps proposed in Chapter 4 and 6 are only possibilities of producing MM UIs. Our methodology allows the introduction of new development sub-steps and/or new transformation systems for realizing sub-steps, thus encouraging the alternative explorations for each sub-step. The introduction of *Synchronization between CICs* (Section 4.4.3.b.3.3) as new sub-step of the transformational approach is a proof of the above statements.

Tool requirements

Requirement 11. Machine processability of involved models: states that the provided ontology should be proposed in a format that can be legible by a machine. This requirement is motivated by the necessity of transposing the ontological concepts into representations that can be processed by machines.

6. Validation

Discussion: This requirement is achieved by the definition of an XML syntax enabling the expression of the concepts of our ontology and in compliance with the graph-based syntax defined for this ontology. The assembly of tools presented in Chapter 5 that manipulate UsiXML format is an evidence of the machine processability of this syntax.

Requirement 12. Support for tool interoperability: refers to the possibility of reusing the output provided by one tool into another. This requirement is motivated by the lack of explicitness of transformations due to their heterogeneous formats that prevents the reuse of transformations outside the context for which they were designed.

Discussion: Support for tool interoperability is positively impacted by a common UI description language that is shared among tools (UsiXML) and by a large coverage of UsiXML in order to accommodate multiple tools. When new concepts corresponding to new interaction modalities need to be introduced in our ontology, the support of new tools can be maintained by relying on *ontological extendibility to new modalities*. However, we are well aware that this requirement is not easy to achieve. As our ontology was continuously evolving in order to better respond to the requirements of the MM UIs, any change entailed the adaptation of the tools resulting in a lot of development effort and delays in the support of modifications brought to the ontology as well. Therefore, coordinating tools in such context is not an easy task.

6.7 Conclusion

In order to conclude this chapter we provide hereafter a set of conclusions issued from the internal validation (i.e., a list of strengths and weakness encountered while developing the case studies and some reflections regarding the empirical validation) and from the external validation (i.e., the extent to which we have addressed the requirements identified at the beginning of this work).

6.7.1 Conclusions issued from the external validation

6.7.1.a Observed advantages

We provide hereafter a set of conclusions regarding the strengths and weaknesses of our methodology by analyzing the three case studies provided in this chapter. Therefore, the strengths encountered during the development process are:

- **Our case studies** showed the **feasibility** of developing a multimodal/monomodal UIs in a principled-based and rigorous manner relying on explicit design options supported by transformation rules gathered in a catalog and selected based on designer's decisions.

- **The diversity of design decisions** highlights the possibility of manipulating UI related artifacts according to different development scenarios and pave the way to consider multiple other alternatives. In particular, new development scenarios can be developed by refinement (e.g. a more elaborated scenario), by composition (e.g., a new scenario by composing several existing scenarios), by transformation (e.g., a newly defined scenario by deriving other forms of scenarios from existing ones) or by reusing.
- **The reuse of transformations.** Two categories were identified:
 - *Highly reusable transformations:* illustrated when transformation systems have been straightforwardly reused from the first case study to the second one and within the same case study for ensuring different tasks. As so, we avoid the development of *ad hoc* transformation rules and enable their capitalization in a consolidated approach while trying to avoid the proliferation of very similar scenarios.
 - *Low reusable transformations:* illustrated in the third case study that required the development of opportunistic rules that apply for very specific design decisions in the context of a reduced number of UIs.

6.7.1.b Observed shortcomings

The weaknesses encountered while realizing these case studies are the following:

- **Lack of expressivity of models.** The fact of decomposing a transformational development process into steps and sub-steps enables an identification of weaknesses of certain models in terms of expressivity. As preciseness in the expression of transformation grows, some models revealed the need of enrichment to allow their exploitation for derivation means. For instance the Task Model had to be expanded with some attributes for a better expression of the modality-independent aspects. Moreover, the Concrete Model had to be enriched with various vocal concepts and synchronization between them and the graphical elements had to be specified. This determined the introduction of a new sub-step in the development process as well.
- **Difficulty in finding an appropriate level of generalization** when defining rules. Conditional graph rewriting offers expressions having no side effect i.e., a rule only affects parts of the graph defined in its scope. Nonetheless, a rule may always have a “wider” scope than planned by its designer. It therefore affects unexpected graph elements. On the other hand, defining very precise rules entails defining a collection of rules for realizing a transformation that could be obtained with the application of one single and more generic rule. An automatic recognition of sets of rules able to be synthesized in one rule would be desirable in this case. This problem is an illustration of the *rule composition* issue raised in the literature.

- **Difficulty in ordering rules within transformation systems.** It happens that two rules of a same transformation system apply to similar graph nodes. These rules are referred in the literature as a *critical pair*. In this case, the ordering of rules has an impact on the graph resulting from the transformation system. *Critical pair analysis* is an algorithmic analysis technique operating on graph grammars and identifying conflicting rule couples. Nonetheless, once these pairs are identified, it remains tedious to modify or re-arrange conflicting rule couples.
- **Difficulty in ordering sub-steps within steps.** In a similar manner to rules, it is not an easy task to order sub-steps within a same step. Each sub-step, along with its associated transformation system, produces a graph presenting certain characteristic i.e., type of nodes and relationships produced during the execution of the sub-step. Arranging sub-steps so that the information produced by the previous sub-step is not modified afterwards remains an undetermined activity. The help of a formal expression of pre- and post- condition of each sub-step would certainly improve this aspect.

6.7.1.c Conclusions of the empirical study

With respect to the empirical study we can conclude that:

- Even if users have a strong preference for multimodality [Ovia99], there is no guarantee that they will issue every command to a system multimodally. For the map browsing application users preferred more the vocal interaction than the MM one as it proved to be more effective. This conclusion is in line with [Ovia97] which observed that user's commands were expressed multimodally 20% of the time, with the rest expressed vocally or graphically.
- Users mix monomodal and multimodal expressions depending on the type of command to convey [Ovia99] and consequently of the widget(s) sustaining it. For multiple selection widgets or single selection in a long list of options the vocal modality was preferred as it proved to be more efficient. On the contrary, for single selection in a group of radio buttons or in a list box the graphical modality was more employed. Therefore, the designers should ensure the flexibility of selecting the appropriate modality.
- Users have a great capacity to adapt to interaction modalities for which they had reduced or even no previous experience. The learning time was significantly improved when the first test was repeated for the vocal modality (Figure 6-39). Therefore, the efforts for encouraging them to experience other interaction modalities than the traditional ones should be enhanced by all means possible.
- The error rate was significantly greater for the vocal modality used alone or combined. This was mainly due to the weakness of the selected technology which requires highly demanding user-system synchronization. Conse-

6. Validation

quently, more emphasize should be put on the development of ergonomic technology supporting MM UIs in order to ensure a more robust HCI.

6.7.2 Conclusions issued from the internal validation

As a conclusion to the discussions offered in the internal validation section, Figure 6-47 provides a subjective estimation of the extent to which we have addressed the requirements identified in the context of this thesis. It can be noticed that these requirements were covered in a great proportion.

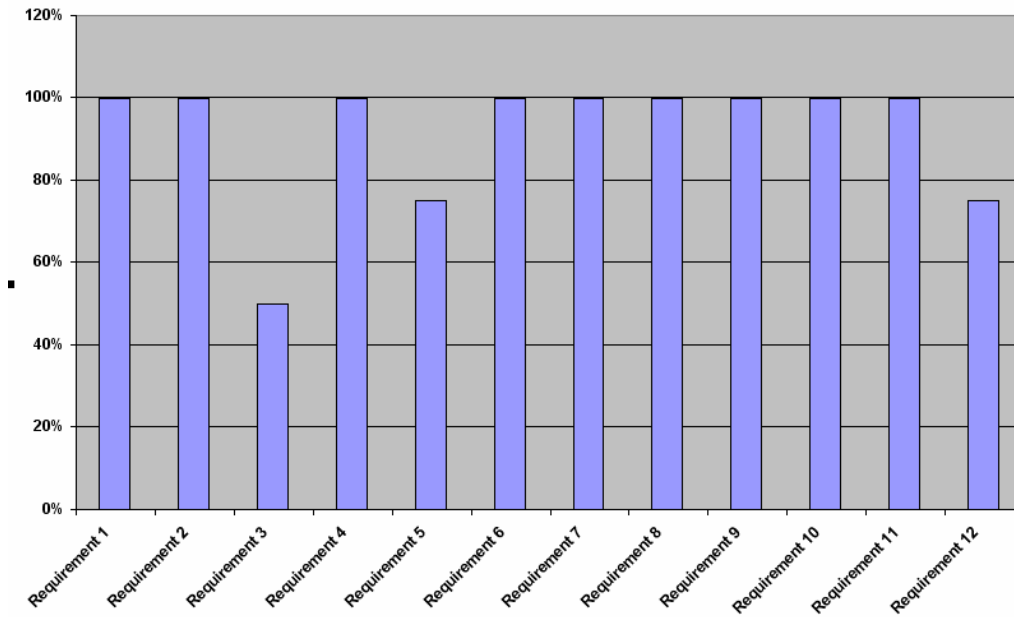


Figure 6-47 Requirements coverage rate

7 Conclusion

7.1 Introduction

This chapter summarizes the contribution brought by the current thesis to the development process of MM UIs with respect to the following aspects: (1) theoretical and conceptual contributions related to the definition, usage and validation of some new original concepts pertaining to the problem, (2) methodological contributions concerned with the methodological guidance provided to UI designers in order to manipulate the newly introduced concepts and (3) tool support expressing how the methodological guidance is supported by a software.

By observing the current state of the art in the field of MM UIs we noticed that most of the development issues tackle the implementation and usability assessment aspects. In addition, a high number of such applications are still developed manually and address very specific problems issued in a particular environment and for a particular category of users. In response, the developers of MM UIs try to provide a circumstantial solution by employing one or more particular interactions. But, these specific solutions are difficult to reuse and modify as they are hard-coded in the implementation. The technologies that support them are often very complex and resource consuming (e.g., time, processing power) which makes them even more difficult to reinstall. Moreover, the existing applications frequently raise extensibility problems when new interaction modalities supported by the constant emergence of novel devices has to be introduced. The existing implementations often address solutions for very complex tasks and are rarely oriented towards information systems.

With respect to these observations, the methodology proposed in the current thesis aimed to cover a greater variety of MM applications accessible for the public at large by providing a general-purpose solution for applications that are not necessarily domain or context-specific.

7.2 Summary of contributions

The contributions of this work can be summarized depending on the aspects composing the methodology:

7.2.1 Theoretical and conceptual contributions

- **Expanded Task Model.** In Section 3.3.1.a we identified a series of shortcomings of the existing Task Model considered for the development of MM applications. In order to better respond to the requirements imposed by such applications, we expanded the model by adding/modifying several attributes along with their values. An extended set of examples involving the newly introduced attributes is offered in order to support the design at the modality independent level.

7. Conclusion

- **Expanded vocal ontology.** In Section 3.4.2.a we identified that the existing ontology suffers from a series of shortcomings with respect to the vocal aspects. Therefore, we reinforced the ontology by expanding it with a set of vocal concepts (Section 3.4.2.b) and relationships (Section 3.4.4.b) between them. They were further formalized in UsiXML language thus enabling to employ them in the transformational process for the development of MM UIs.
- **Structuring a multimodal instruction.** In software engineering the structure of any ordinary instruction is composed of three atomic elements: the action, the object over which it applies and the parameters characterizing the objects. A MM instruction is submitted to the same observation, but introduces a new variable in the equation: the interaction modality(s) employed for the specification of each element composing the instruction. Moreover, the need of identifying the possible combinations of elements in terms of their cardinality would be a benefit for the process of selecting the appropriate interaction modality. Consequently, Section 3.3.3 identified the attributes that support the structure of a MM instruction (i.e., *actionType* and *actionItem*). Furthermore, four possible general cases of combinations between the attributes were identified based on their cardinality. As these attributes are placed in our ontology at the abstract level (i.e., modality-independent), they enable the designers to choose the most suitable interaction modality according to their values and purpose.
- **Synchronization between modalities.** In MM UIs the synchronization represents a key aspect as it enables to correlate the data received/sent from one modality to another. In order to support this requirement we introduced in Section 3.4.4 the synchronization relationship between graphicalCIOs and vocalCIOs with four particular instantiations.
- **Stylistics for vocal CIOs.** The need of facilitating the understanding and the manipulation of vocal objects employed in software tools required the introduction of a graphical representation associated to some of the introduced vocal concept (Section 3.6).

7.2.2 Methodological contribution

- **Design space.** The existing development of MM applications involves a manual process for the generation of UIs and it does not take into consideration any methodology based on design options. Our work defines a design space (Section 4.2) composed of design options that govern design rules encoded as graph grammars which are automated in order to ease the development life cycle of MM UIs. The advantages provided by this approach are three-fold: (1) all design options are documented and allow summarizing any design in terms of design options values, (2) several different designs of MM UIs may be compared according to the design options in order to assess the design quality in terms of factors, such as utility, usability, portability and (3) the design space allows to discover potential new design option values or to introduce new design options assigned to yet under explored design features which could have a positive impact over the facility of development and quality ergonomics of MM applications.

7. Conclusion

The number of design options composing our design space is the result of a trade-off made in order to ensure:

- **Low treshold:** a high number of design options ensuring a very high level of coverage with respect to the design features of the UIs to develop would prevent designers from a rapid understanding of the manner in which they are defined, justified and employed. Contrary to the design space rationale, the design workload would be increased.
- **High ceiling:** a reduced number of design options would impose limitations over the features supported by the generated UIs.
- **Wide walls:** a few number of design options would prevent designers from addressing a wide range of explorations with respect to the type of UIs to produce.

An important aspect concerning the design space is the identification of dependencies between the composing design options. A design space is said to be orthogonal if all dimensions are independent of each other. Even if we would like to define an orthogonal design space, this condition is not fulfilled as dependencies between different design options have been identified. So far, we have observed the dependencies illustrated in Table 7-1. The first column specifies the design option values creating the dependencies and the second column the design option values determined by the dependency.

Design option value causing the coupling	Coupled design option value
Sub-task presentation = separated	Navigation type containment = local
Sub-task presentation = separated	Control type containment = local
Navigation & control type = combined +	Control type containment = local
Navigation type containment = local	
Navigation & control type = combined +	Navigation type containment = local
Control type containment = local	
Concretization of navigation & control = combined +	Control type containment = global
Navigation type containment = global	
Navigation & control type = combined +	Navigation type containment = global
Control concretization placement = global	
Navigation & control type = combined +	Control type cardinality = multiple
Navigation type cardinality = multiple	
Navigation & control type = combined +	Navigation type cardinality = multiple
Control type cardinality = multiple	
Input = graphic (A)	Immediate feedback ≠ vocal (A)
Input = multimodal (E,C,R)	Immediate feedback ≠ vocal (A)

Table 7-1 Dependencies between design options

7. Conclusion

- **Expanded model-to-model transformational approach:**

The current thesis introduces in Section 4.3.4 the *color* as a new feature of our transformational approach. Thanks to it, we are able to endow the concepts of our ontology with more semantic that will progressively enable us to support this approach with improved concepts and operations defined between them:

 - **Colored concepts.** In order to distinguish between the modality independent/dependent character of the concepts pertaining to our ontology, colors have been considered (Section 4.3.4.a.1). Thus, we chose a neutral color (i.e., black) in order to represent the independent concepts, whereas a specific non-neutral color is associated to each modality (i.e., red for the graphical concepts and blue for the vocal ones).
 - **Colored graphs.** We expanded the abstract syntax of our transformational approach by introducing the notion of colored graphs (Section 4.3.4.a.2) thanks to a pair of functions that attach to each node and edge of a graph a particular color. Furthermore, two operations over colored graphs have been defined: merging and splitting (Section 4.3.4.a.3).
 - **Colored transformation rules.** Thanks to the aforementioned concepts, we were able to define (Section 4.3.4.a.5) the notion of colored transformation rule (with two particular concretizations: monocolored or multicolored) and to introduce two operations that apply over them (i.e., merging and splitting). Consequently, we will benefit from a series of advantages in terms of flexibility of their application. Thus, having at hand a multicolored rule, the generation of an application that enables a particular type of interaction (i.e., monomodal or multimodal) is done by considering the corresponding colors. Moreover, multiple monomodal transformation rules supporting particular interaction modalities can be merged together in order to obtain MM rules that ensure the generation of MM UIs. Another advantage consists of the fact that the designers' decision of generating a particular type of application can be supported by a tool that could automatically processes the suitable color(s) as feature(s) of colored transformation rules.
- **Transformation rule catalog – support for design space.** The graph-based transformation rules that support our design space were gathered in a complete and systematic structure: a transformation rule catalog. Based on it, designers could manually select the corresponding transformation rule(s) of their design decisions, thanks to the mapping specified between each design value and transformation rules (Section 4.3.5). The transformation rules provided by our transformation catalog are hard to implement and apply, thus being resource consuming in terms of time to learn to design such rules, time to apply them, etc. However, the way in which they are structured in transformation systems and development steps allow designers to reuse their organizational logic for a further different implementation.
- **New methodological sub-step.** [Limb04] identifies the development sub-steps for the generation of graphical and vocal UIs. We reuse these sub-steps for MM UIs but we also add a new one, *synchronization between CICs*, in order to support the methodological development process of MM UIs. The newly introduced sub-step

7. Conclusion

ensures the coordination of vocalCIOs and the graphicalCIOs by employing the synchronization relationship.

7.2.3 Tools developed

As the design space introduced in the current dissertation offers the advantage of being independent of any tool, any developer of MM UIs could take advantage of an explicit support of the introduced design options. Consequently we considered MultimodalXML, an assembly of five software modules synchronized between each other that aim to reduce the designers' set of concerns by limiting the amount of design decisions he could take.

7.3 Future work in prospect

With respect to some shortcomings observed in Section 6.7.1.b future work could address the following concerns:

- **Context of use and domain considerations - building a knowledge base of inference rules that recommends the appropriate design option value to select.** An important aspect to consider when developing MM UIs is the context in which these applications will be used, as well as the domain they cover. Depending on the three parameters characterizing the context, different design decisions could be imagined. Intuitively, a MM UI run on a desktop or notebook PC would consider a presentation where all tasks are conveyed all at once in grouped lists, whereas on a PDA platform a separated presentation would be more convenient. A mobile platform used in a noisy environment would better support a UI that enables graphical input/output modalities, whereas used in quiet contexts where users have eyes-busy a vocal input/output interaction would be more suitable. Moreover, for users employing the system in a domain where the task criticality is high (i.e., air traffic control) a confirmation of the previously input would be recommended, whereas for users with a high device and system experience achieving a less complex task (i.e., search the translation of a word in an online dictionary) confirmation wouldn't be required. But these intuitions suppose an empirical validation based on which a set of inference rules recommending appropriate design option values to select could be derived and gathered in a knowledge base.
- **Extend the methodology to support development of context-aware systems.** Our methodology addresses the development of MM UIs for predefined and constant contexts of use that do not support any dynamic run-time migration from one modality to another. However, nowadays, the interaction has to be adapted to different situations and to a context in constant evolution [Calv03]. This diversity of interaction contexts emphasizes the complexity of MM system design for which solutions should be provided in order to enable systems to have run-time context-aware capabilities [Sott07, Vand08]. Therefore, we would like to analyze a solution that considers a set of selection rules based on which different CUIs are generated from the same AUI depending of several context parameters. Such an approach was applied and described in [Rous05], where a behavioral model formalized by a base of election

7. Conclusion

rules allows the selection of the most suitable output modality. This approach could be extended to input modalities as well.

- **Design space improvement.** We may want to perform the following activities over the design space defined so far: (1) reduce the semi-dependent dimensions; (2) introduce more values for each design option; (3) introduce new dimensions while maximizing the independency between them.
- **Extend the methodology to support new interactions.** More and more UIs are supporting nowadays new modalities of interaction. Hereafter, we provide our point of view over the extension process of the methodology when a new interaction should be considered:
 - **Ontological extension.** We will put into balance the necessity of introducing new concepts and the possibility to adapt the existing ones in order to support the new interaction. For instance, the introduction of the tactile interaction will probably consist only in some physical constraints imposed on the existing graphical objects. A decision that will consider both aspects could be considered as well.
 - **Method extension.** This extension will consider two aspects:
 - (1) The design space: first we will analyze if the existing design options still keep their modality-independent character with respect to the newly introduced modality. Second, we will study the opportunity of introducing new design option values or new design options that could provide more guidance to the designer when considering the new interaction alone or combined. The validity of these new options with respect to the concretization of the already existing interactions should be checked as well.
 - (2) The transformational method: first, we will analyze if the systematic development approach proposed in this thesis still keeps its coherence when confronted with the new interaction. Second, we will examine the necessity of introducing new development sub-steps and consequently adapting/extending the existing transformational rules gathered in the catalog in order to support the new interaction. The objects over which these rules are applied will be assigned with a new color in order to benefit from the advantages provided by the multicolored rules.
 - **Tool extension.** Depending on the extensions operated at the ontological and methodological levels the supporting software should be adapted accordingly.
- **Redesign a brand new UsiXML software support.** The purpose of our design space is to guide the designer during the development life cycle when having to decide between different design alternatives. In line with this goal, a new software concretized in a design space-based tool (Figure 5-12) could be developed. It would enable to select design values hiding transformational rules, thus absolving designers from useless and workloading details. Moreover, it would be useful to present a graphical preview of the design decision outcome. Thus, a clearer picture of the presentation (supported by the introduced stylistics) and behavior of the future UIs could be provided.
- **Analyse the possible extensions of the colored transformation rules.** We would

7. Conclusion

like to analyze whether the introduction of color as a new feature of our ontology is a conservative extension with respect to the graph grammar properties such as termination, confluence, parallel and sequential independence.

- **Usability evaluation of transformation rules.** The aim is to assess the level of usability covered by applying a set of transformation rules to develop MM UIs. Specifically, we are interested in answering the following questions: Based on a set of already identified usability criteria for the evaluation of HCI, what is the level of coverage of the transformation rules? Are all usability criteria covered? What is the level of coverage by modality? What is the level of coverage by design option? What is the level of coverage by transformational level? Which criteria are preserved in all the transformation levels? Could we generalize this reasoning for a general MDA approach involving UI development? The ultimate goal would be to investigate whether MDA-compliant methods ensure a guaranteed level of usability through model transformations.
- **Evaluation of the MM UI usability based on cognitive psychology principles.** Decision making, as a feature of the cognitive psychology, plays an important role in the area of HCI by creating the context for defining major design options for information systems in order to pave the way to a structured development life cycle. [Mars87] specifies a set of summary qualitative principles derived by usability psychologists from cognitive psychology and designed to offer detailed guidance for designers during the development process. Whereas the cognitive psychology offers a support for the usability of the decision making, the MM UI design studies published so far in the area of Software engineering and HCI are surprisingly rare. Furthermore, there are few ongoing works on usability of MM UIs mainly because there are not so many MM applications. There have been a number of studies (e.g., [Lars03a]) of the way designers should conceptualize their MM UIs, but these give little insight into the way design options are formulated or decisions should be actually reached. Thus, the usability of the design options for MM applications still remains an uncovered research area as there are no usability MM applications experiments for this. As so far we focused on the feasibility of code generation, the next steps will consider usability experimental studies that will take into account qualitative principles derived from cognitive psychology and their applicability to the design space.

7.4 Some personal concerns

This section presents some personal reflections and the resulting concerns over the current dissertation with respect to the connection between the effort made to realize it and the obtained results.

We base our analyse on the *Pareto's principle* which has its roots in an observation made over the Italian people stating that: *80% of Italy's wealth was produced by 20% of the population*. This principle has been further validated in other areas of expertise thanks to numerous empirical studies, thus giving rise to the so called *80 – 20 Rule*. For instance, supermarkets noticed that 80% of their stock comes from 20% of their suppliers. Also

7. Conclusion

80% of the production is produced by 20% of the company staff and 80% of their problems are caused by 20% of the staff. Consequently, the rule practically states that 80% of the problem is solved with only 20% of effort. The corollary of this rule is that for the remaining 20% of the problem, 80% of the effort would be required.

Therefore, we analyzed the extent to which this rule is validated by the methodology employed in the current research in order to solve the problem of MM UI development. For this purpose we discuss two points of view:

1. *The author's point of view:* we examined first the three dimensions of our methodology (i.e., models, method, tools) by assigning weights depending on their contribution to the final outcome (Figure 7-1). Thus, even subjective, we consider that the introduced ontology and the method manipulating its elements have a greater rate (i.e., around 80 %) than the software support (i.e., around 20%) in the total outcome of the methodology. This is mainly due to the fact that the introduced ontology, the design options composing the design space and the highly structured development process guided by these options are independent of any implementation or tool support thus providing an output that could be useful to any designer of MM UIs. In a second step we estimated the effort made during this research in order to reach the outcomes produced by each of the methodological dimensions. We conclude that most of these efforts were dedicated to the implementation aspects (i.e., developing the modules of the MultimodalXML tool, designing and manually specifying the transformation, manually applying the XSL Transformations) which proved to be a highly time and resource-consuming activity.

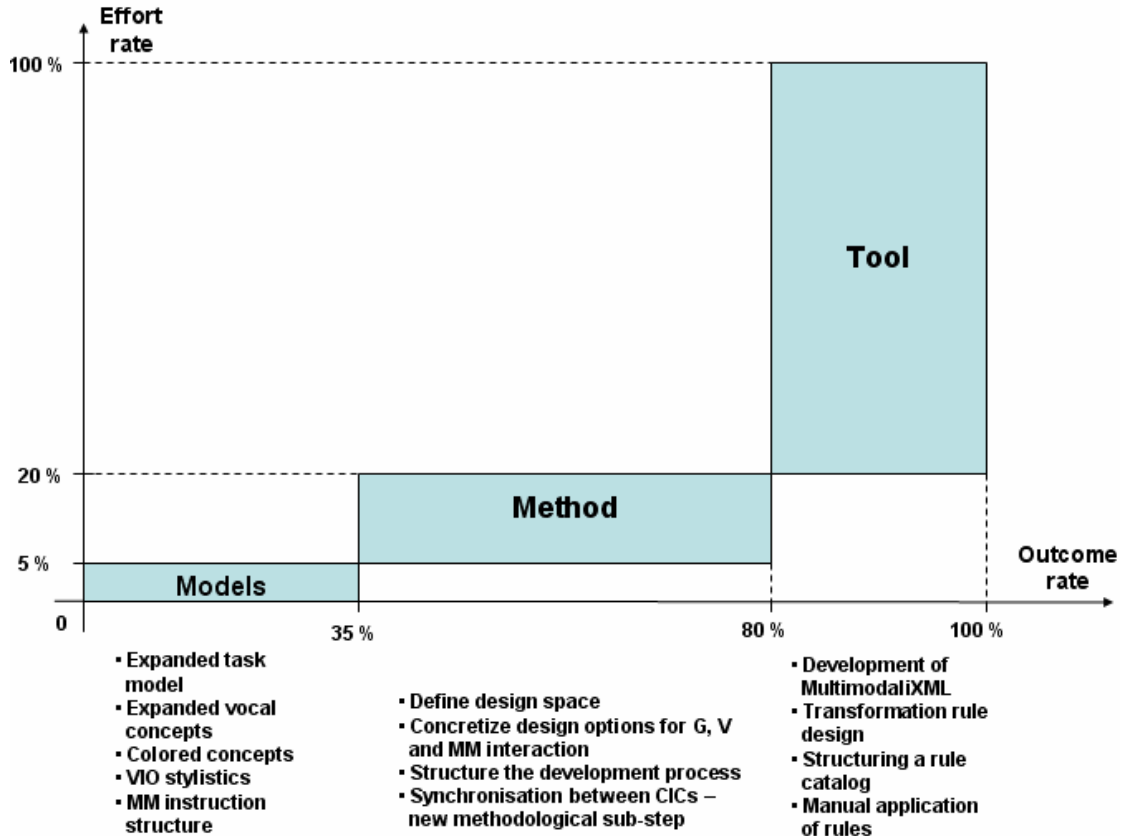


Figure 7-1 Connection between the effort rate and the outcome rate of our methodology

7. Conclusion

2. *The extern designer point of view:* If we were to provide the methodology developed in this thesis to a designer in order to apply it on a different case study than those presented in Chapter 6, we consider that the introduced ontology and the method manipulating it should remain unchanged thanks to their independence of the technology. On the other hand, the transformational approach and the tools implementing it should be redesigned as the transformations rules proved to be difficult to manipulate and apply, requiring a high threshold and sometimes the development of opportunistic rules that can be applied only for several cases.

As a result of these discussions we conclude that the effort involved in providing a valuable methodology for the development of MM UIs and the outcome produced by it are in line with the 80 – 20 Rule. We have identified the 80% of the results for which the effort involved in the activities producing them was relatively low. As suggested by several analysts of 80 – 20 Rule from now on the focus should be set on the rest of the 20% of results to produce (in our context, the implementation aspects) for which an 80% effort rate is required. In line with this observation we consider that a design space-based tool as the one illustrated in Figure 5-12 could help designers minimize the development effort when building MM applications.

7.5 Concluding remarks

This thesis introduces a methodology for the development of MM UIs that applies a transformational approach over a set of models thanks to transformation rules employed in different development steps in order to offer guidance for coding complete MM applications. The methodology (as defined in Section 1.4) is delineated by a set of requirements that are elicited and motivated by the state of the art presented in Chapter 2. The validation of the proposed methodology is achieved by applying it over the case studies presented in Chapter 6. Their main goal is to show the feasibility of the methodology and provide designers with some explicit guidance on what to decide for their future UI, while exploring various design alternatives.

The diversity of the UIs that have been developed based on the design space highlights the possibility of manipulating related UIs and paves the way to consider multiple other alternatives. In particular, new types of UIs can be developed *by refinement* (e.g., more elaborated UIs obtained by taking into consideration more design option values), *by composition* (e.g., new types of UIs obtained by combining several existing design option values), *by transformation* (e.g., new UIs obtained by deriving existing UIs based on the modifications made over the design option values) or by reusing. The possibility of reusing already developed UIs is a consequence of the reusability feature of the transformations rules. This feature has been demonstrated straightforwardly when applied from one case study to another. Thus, we avoid the development of transformation catalogs that are applied only for a particular case study and consequently we prevent the proliferation of similar UIs.

7. Conclusion

References

A

- [Abra99] Abrams, M., Phanouriou, C., Batongbacal, A.L., Williams, S., Shuster, J., *UIML: An Appliance-Independent XML User Interface Language*, in Mendelzon, A. (Ed.), Proceedings of 8th International World-Wide Web Conference WWW'8 (Toronto, May 11-14, 1999), Elsevier Science Publishers, Amsterdam, 1999. Available online: <http://www8.org/w8-papers/5b-hypertext-media/uiml/uiml.html>.
- [Abra04] Abrams, M., Helms, J., *User Interface Markup Language (UIML) Specification Working Draft 3.1*, 11 March 2004. Available online: <http://www.oasis-open.org/committees/download.php/5937/uiml-core-3.1-draft-01-20040311.pdf>.
- [Agra03] Agrawal, A., *Metamodel Based Model Transformation Language*, Proceedings of ACM International Conference on Object-Oriented Programming Systems, Languages and Applications OOPSLA'2003, (Anaheim, October 26-30, 2003), ACM Press, New York, 2003, pp. 386-387.
- [Albr83] Albrecht, A., J., Gaffney, J., E., *Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation*, IEEE Trans. Software Engineering 9(6), 1983, pp. 639-648.
- [Alle83] Allen, J., F., *Maintaining Knowledge about Temporal Intervals*, Communications of the ACM, 26(11), 1983, pp. 832-843.
- [Aneg04] Aneg H., Niklfeld G., et al., *Multimodal Interfaces in Mobile Devices – The MONA Project*, Proc. of the Emerging Applications for Wireless and Mobile Access Workshop MobEA'II (New York, May 18, 2004).
- [Awde06] Awde, A., Hina, M., D., Tadj, C., Ramdane-Cherif, A., Bellik, Y., *Information Access in a Multimodal Multimedia Computing System for Mobile Visually-Impaired Users*, Industrial Electronics, 2006 IEEE International Symposium, Volume4, ISBN 1-4244-0496-7, 9-13 July 2006, pp. 2834-2839.

B

- [Barn08] Barnett, J., et al., *State Chart XML (SCXML): State Machine Notation for Control Abstraction*, W3C Working Draft, 16 May 2008. Available online: <http://www.w3.org/TR/2008/WD-scxml-20080516>.
- [Bass91] Bass, L., Pellegrino, R., Reed, S., Seacord, R., Sheppard, R., Szezur, M., R., *The Arch model: Seeheim revisited*, User Interface Developer's workshop version 1.0, 1991.
- [Bast97] Bastien, J., M., C., Scapin, D., L., *Ergonomic criteria for evaluating the ergonomic quality of interactive systems*, Behaviour and Information Technology 16, pp 220-231.
- [Beau00] Beaudouin-Lafon, M., *Instrumental Interaction: An Interaction Model for Designing Post-WIMP User Interfaces*, Proceedings of the ACM International Conference on Human Factors in Computing Systems CHI'2000 (The Hague, April 1-6, 2000), ACM Press, New York, 2000, pp. 446-453.
- [Bern06] Bernsen, N., O., Dybkjaer, L., *Multimodal Usability MULUS*, SIMILAR Multimodal Usability BOOK, 18.08.2006.
- [Bert05] Berti, S., Paterno, F., *Migratory MultiModal interfaces in MultiDevice environments*, Proceedings of the 7th International Conference on Multimodal Interfaces, ICMP'2005 (Trento, 4-6 October, 2005), ACM Press, New York, 2005, pp.92-99.
- [Bell92] Bellik, Y., Teil, D., *Les types de multimodalités*, Actes des 4^{èmes} Journées sur l'Ingénierie des Interfaces

References

- Homme-Machine IHM'92, Paris, 1992.
- [Blan06] Blanquet, J., Gatefin, J., Cherrier, P., *Recommandations ergonomiques, Projet VERBATIM, Sous-projet 1 – Lot 2, version 2.4*, 2006.
- [Boda94] Bodart, F., Vanderdonckt, J., *On the Problem of Selecting Interaction Objects*, Proceedings of BCS Conference HCI'94 "People and Computers IX" (Glasgow, 23-26 August 1994), G. Cockton, S.W. Draper, G.R.S. Weir (eds.), Cambridge University Press, Cambridge, 1994, pp. 163-178.
- [Bodn04] Bodnar, A., Corbett, R., Nekrasovski, D., *AROMA: Ambient awaReness through Olfaction in a Messaging Application - Does Olfactory Notification Make 'Scents'*, Proceedings of ACM International Conference of Multimodal Interfaces ICMI'2004 (State College, October 13-15, 2004), ACM Press, New York, 2004, pp. 183-190.
- [Bolt80] Bolt, R., A., *Put-that-there: Voice and gesture at the graphics interface*, Proceedings of the 7th Annual Conference on Computer Graphics and Interactive Techniques SIGGRAPH'80 (Seattle, 1980), pp. 262-270.
- [Bouc04a] Bouchet, J., Nigay, L., Ganille, T., *ICARE Software Components for Rapidly Developing Multimodal Interfaces*, Proceedings of the 6th ACM International Conference on Multimodal Interfaces ICMI'2004 (State College, 2004), ACM Press, New York, 2004, pp. 251-258
- [Bouc06] Bouchet, J., *Ingénierie de l'interaction multimodale en entrée: Approche à composants ICARE*, Ph.D. thesis, Université Joseph Fourier, Grenoble, December 7th, 2006.
- [Brew96] Brewster, S., A., McGookin, D., K., Miller, C., A., *Olfoto: Designing a Smell-Based Interaction*, Proceedings of ACM International Conference on Human Aspects in Computing Systems CHI'2006 (Montréal, April 22-27, 2006), ACM Press, New York, 2006, pp. 653 - 662.
- [Bygs07] Bygstad, B., Ghinea, G., Brevik, E., *Systems Development Methods and Usability in Norway: An Industrial Perspective*, Usability and Internationalization, HCI and Culture, Lecture Notes in Computer Science, Springer Berlin, Volume 4559, 2007, ISBN 978-3-540-73286-0, August 24, 2007, pp. 258-266.
- ## C
- [Calv03] Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., Vanderdonckt, J., *A Unifying Reference Framework for Multi-Target User Interfaces*, *Interacting with Computers*, 15(3), June 2003, pp. 289-308.
- [Clar99] Clark, J., *XSL Transformations (XSLT)*, Version 1.0, W3C Recommendation 16 November 1999. Available online: <http://www.w3.org/TR/xslt>.
- [Cohe98] Cohen, P., R., Johnston, M., *The Efficiency of Multimodal Interaction: A Case Study*, Proceedings of the International Conference on Spoken Language Processing ICSLP'98 (Darling Harbour, 1998), pp. 249-252.
- [Conk88] Conklin, J., Begeman, M., L., *gIBIS: A Hypertext Tool for Exploratory Policy Discussion*, *ACM Transactions on Office Information Systems*, 6(4), 1988, pp. 303-331.
- [Cole85] Cole, I., Lansdale, M., Christie, B., *Dialogue design guidelines*, *Human Factors of Information Technology in the Office*, Christies, B. (Ed.), John Wiley, Chichester, 1985.
- [Cout92] Coutaz, J., *Multimedia and Multimodal User Interfaces: A Software Engineering Perspective*, Proceedings of the East-West International Conference on Human-Computer Interaction EWHCI'92 (St. Petesburg, 4-8 August, 1992).
- [Cout95] Coutaz, J., Nigay, L., Salber, D., Blanford, A., May, J., Young, R.M., *Four easy pieces for assessing the usability of multimodal interaction: the CARE properties*, Proceedings of 5th IFIP TC 13 International Conference on Human-Computer Interaction INTERACT'95 (Lillehammer, 27-29 June 1995),

References

- Nordbyn, K., Helmersen, P.H., Gilmore, D.J., Arnesen, S.A. (Eds.), Chapman & Hall, London, 1995, pp. 115–120.
- [Crea00] Crease M., Brewster S., A., Gray P., *Caring, sharing widgets: a toolkit of sensitive widget*, Proceedings of 14th Annual Conference of the British HCI Group (5-8 September 2000), British Computer Society conference series, Sunderland, England, pp. 257-270.
- [Czar03] Czarnecki, K., Helsen, S., *Classification of Model Transformation Approaches*, Proceedings of the OOPSLA'03 Workshop on Generative Techniques in the Context of Model-Driven Architecture, anaheim, October 26-30, 2003.
- D**
- [Debo06] De Boeck, J., Raymaekers, C., Conix, K., *Comparing NiMMiT and Data-Driven Notations for Describing Multimodal Interaction*, Proceedings of 5th International Workshop on Task Models and Diagrams for User Interface Design TAMODIA'2006, (Hasselt, Belgium, October 23-24, 2006), Lecture Notes in Computer Science 4385 Springer 2007, ISBN 978-3-540-70815-5, pp. 139-146.
- [Deva91] Devauchelle, P., *User-friendly recommendations for voice services designers*, France Telecom.
- [Dijk76] Dijkstra, E. W., *The discipline of programming*, Prentice Hall, Engelwood Cliffs, NJ, 1976.
- E**
- [Ehri99] Ehrig, H., Engels, G., Kreowski, H-J., Rozenberg, G., *Handbook of Graph Grammars and Computing by Graph Transformation, Application, Languages and Tools*, Vol. 2, The Graph Transformation Language AGG, World Scientific, Singapore, 1999.
- [Enge89] Engelbeck, G., Roberts, T., *The effects of several voice-menu characteristics on menu selection performance*, Technical report ST0401, US West Advanced Technologies, Englewood, CO.
- G**
- [Gait07] Gaitanis, K., Vybornova M., O., Gemo, M., Macq, B., *Multimodal High Level Fusion of Input Commands as a Semantic Goal-Oriented Cooperative Process*, Proceedings of 12th International Conference Speech and Computer, SPECOM, 2007.
- [Gonz06] Gonzalez C., J., M., *A Method for Developing 3D User Interfaces for Information Systems*, DEA Thesis, UCL, Louvain-la-Neuve, June 2006.
- [Goul87] Gould, J., D., Boies, S., J., *The 1984 olympic message system: A test of behaviour principles of system design*, Communications of the ACM, 30, pp. 758-769.
- [Gram96] Gram, C., Cockton, G., *Design Principles for Interactive Software*, IFIP's Working Group 2.7, First edition 1996, Chapman and Hall, ISBN 0-412-72470-7.
- [Grub90] Gruber, T., R., Russell, D., M., *Design Knowledge and Design Rationale: A Framework for Representation, Capture and Use*, Knowledge Systems Laboratory, Stanford University, 1990, KSL 90-45.
- H**
- [Hewe96] Hewett, T., Baecker, R., Carey, T., Gasen, J., Mantei, M., Perlman, G., Strong, G., Verplank, W., *Curricula for human-computer interaction*, Technical Report 608920, ACM Special Interest Group on Computer-Human Interaction Curriculum Development, 1996.
- [Hoov91] Hoover, S., Rinderle, J., *Models and Abstractions in Design. Design Studies*, Volume 12, Number 4, October, 1991.

References

[Hura03]

Hura, S., L., Owens, R., *The truth about multimodal interaction*, August 2007. Available online: http://www.microsoft.com/speech/docs/Intervoice_Multimodal_Article.htm.

I

[IBM93]

IBM, *Object-Oriented Interface Design, IBM Common User Access Guidelines*, IBM document, Que publishing, March 1993.

[IBM03a]

IBM, *WebSphere Voice Server for Multiplatforms, VoiceXML Programmer's Guide*, Version 4.2, September 2003.

[IBM03b]

IBM White Paper, *Multimodal Application Design Issues*, December 2003. Available online: http://www.ibm.com/developerworks/websphere/library/techarticles/0312_li/0312_li.html#download.

[IBM05]

IBM, *WebSphere Voice Toolkit Getting Started Version 6.0., Second Edition, November, 2005*. Available online: <http://publib.boulder.ibm.com/infocenter/pvcvoice/51x/index.jsp?topic=/com.ibm.voicetools.callflow.doc/ccfpalette.html>.

[IEEE90]

IEEE society, *Glossary of Software Engineering Terminology*, IEEE Standard #610.12-1990, IEEE press, 1990.

J

[Java98]

Java, *Java Speech API Programmer's Guide*, Version 1.0, October 26, 1998, Sun Microsystems. Available online: <http://java.sun.com/products/java-media/speech/forDevelopers/jsapi-guide/UserInterface.html>.

[Jens98]

Jensen, K., *A brief introduction to colored Petri nets*, Proceedings of the Workshop on the Applicability of Formal Models (Aarhus, Denmark, 2 June 1998), pp. 55-58.

[John95]

Johnsgard, T., J., Page, S., R., Wilson, R., D., Zeno, R., J., *A Comparison of Graphical User Interface Widgets for Various Tasks*, Proceedings of the Human Factors & Ergonomics Society - 39th Annual Meeting, Human Factors and Ergonomics Society, October 1995., pp. 287-291.

K

[Kaye04]

Kaye, J., *Making scents: Aromatic output for Human-Computer Interaction*, Interactions, 11(1), 2004.

[Kats03]

Katsurada, K., Nakamura, Y., Yamada, H., Nitta, T., *XISL :A Language for Describing Multimodal Interaction Scenarios*, Proceedings of the 5th International Conference on Multimodal Interfaces ICMI'03 (Vancouver, Canada, 2003), ACM Press, New York, 2003, pp.281-284.

[Kawa96]

Kawai S., Aida H., Saito T., *Designing interface toolkit with dynamic selectable modality*, Proceedings of the 2nd Annual ACM Conference on Assistive Technologies Assets '96 (Vancouver, British Columbia, Canada, April 11 - 12, 1996), ACM Press, New York, NY, pp. 72-79.

[Kawa03]

Kawamoto, S., Shimodaira, H., Sagayama, S., et al., *Galatea: Open-Source Software for Developing Anthropomorphic Spoken Dialog Agents*, Life-Like Characters, Tools, Affective Functions and Applications, Helmut Prendinger et al. (Eds.) Springer, November 2003, pp. 187-212.

[Klem00]

Klemmer, S., R., *SUEDE: A Wizard of Oz Prototyping Tool for Speech User Interfaces*, in CHI Letters, Proceedings of 13th Annual ACM Symposium on User Interface Software and Technology UIST'2000, pp. 1-10.

[Knol90]

Knolls, C., *Voice Messaging User Interface Forum*, Specification document, April, 1990.

References

L

- [Laca05]
Lacaz, X., *Conception rationalisée pour les systèmes interactifs, Une notation semi formelle et un environnement d'édition pour une modélisation des alternatives de conception*, PhD Thesis, 20 June 2005. Available online: <http://liihs.irit.fr/lacaze>.
- [Lars03a]
Larson, J., A., *Commonsense Guidelines for Developing Multimodal User Interfaces*, Larson Technical Services, 3 April, 2003. Available online: <http://www.larson-tech.com/MMGuide.html>.
- [Lars03b]
Larson, J., A., Raman, T., V., Raggett, D., *W3C Multimodal Interaction Framework*, W3C Note 6 May 2003. Available online: <http://www.w3.org/TR/mmi-framework>.
- [Lars06]
Larson, J., Intel, *Common Sense Suggestions for Developing Multimodal User Interfaces*, W3C Working Group Note 11 September, 2006. Available online: <http://www.w3.org/TR/mmi-suggestions>.
- [Lew95]
Lewis J., R., *Computer Usability Satisfaction Questionnaires: Psychometric Evaluation and Instructions for use*, International Journal of Human-Computer Interaction 7(1), 1995, pp. 57-78.
- [Limb00]
Limbourg, Q., Vanderdonckt, J., Souchon, N., *The Task-Dialog and Task-Presentation Mapping Problem: Some Preliminary Results*, Proceedings of 7th International Workshop on Design, Specification, Verification of Interactive Systems DSV-IS'2000 (Limerick, 5-6 June 2000), F. Paternò, Ph. Palanque (eds.), Lecture Notes in Computer Science, Vol. 1946, Springer-Verlag, Berlin, 2000, pp. 227-246.
- [Limb04a]
Limbourg, Q., Vanderdonckt, J., *Transformational Development of User Interfaces with Graph Transformations*, Proceedings of 5th International Conference on Computer-Aided Design of User Interfaces CADUI'2004 (Madeira, 14-16 January, 2004), Kluwer Academics Publishers, Dordrecht, 2004.
- [Limb04b]
Limbourg, Q., *Multi-Path Development of User Interfaces*, PhD thesis, University of Louvain, November, 2004.
- [Löwe93]
Löwe M., *Algebraic approach to single-pushout graph transformation in Theoretical Computer Science*, Vol. 1, 1993, pp. 181-224.
- ### M
- [Mac89]
MacLean, A., Young, R., Moran, T., *Design rationale: the argument behind the artifact*, Proceedings of ACM Conference on Human Aspects in Computing Systems CHI'89 (Austin, 30 April - 4 May 1989), ACM Press, New York, 1989, pp. 247-252.
- [Mac91]
MacLean, A., Young, R., Bellotti, V., Moran, T., *Questions, Options and Criteria: Elements of Design Space Analysis*, Lawrence Erlbaum Associates, 1991, pp. 201-250.
- [Maes03]
Maes, S., H., Saraswat, V., *Multimodal Interaction Requirements*, W3C Note 8 January 2003. Available online: <http://www.w3.org/TR/mmi-reqs/#Inputmodalityrequirements>.
- [Mars87]
Marshall C., Nelson C., Gardiner M., *Design guidelines. In Applying Cognitive Psychology to User- Interface Design*, M. M. Gardiner and B. Christie (eds), Chichester, Wiley & Sons Ltd, 1987.
- [Mart02]
Martin, J.-C., Kipp, M., *Annotating and Measuring Multimodal Behaviour - Tycoon Metrics in the Anvil Tool*, Proceedings of 3rd International Conference on Language Resources and Evaluation LREC'2002 (Las Palmas, Canary Islands, Spain, 29-31 May 2002).
- [Mart01]
Martin, J., C., Grimard, S., Alexandri, K., *On the annotation of the multimodal behavior and computation of cooperation between modalities*, Proceedings of the workshop on Representing, Annotating, and Evaluating Non-Verbal and Verbal Communicative Acts to Achieve Contextual Embodied Agents (May 29, 2001, Montreal) in conjunction with the 5th International Conference on Autonomous Agents, pp 1-7.

References

- [Mart97]
Martin, J., C., *TYCOON: Theoretical Framework and Software Tools for Multimodal Interfaces*, Intelligence and Multimodality in Multimedia Interfaces, AAAI Press, 1997.
- [McGe98]
McGee, D., R., Cohen, P., R., Oviatt, S., *Confirmation in multimodal systems*, Proceedings of 36th annual meeting on Association for Computational Linguistics, ACL, 1998, pp. 823 – 829.
- [Medi07]
Medina, J-L., Chessa, S., Front, A., *A Survey of Model Driven Engineering Tools for User Interface Design*, Proceedings of 6th International Workshop on Task Models and Diagrams TAMODIA'2007 (November 7-9, 2007), Springer, Berlin, pp. 84–97.
- [Mell03]
Mellor S., J., Clark A. J., *Introduction to Model Driven-Development*, IEEE Software 20(5), 2003, pp. 14-18.
- [Mens06]
Mens, T., Van Gorp, P., *A Taxonomy of Model Transformation*, Proceedings of International Workshop on Graph and Model Transformation GraMoT'2005, Electronic Notes in Theoretical Computer Science, 152 (2005), pp. 125–142.
- [Meri06]
Merisol, M., Badia, F., *Evaluation de la maquette d'un service multimodal de recherché d'itinéraire dans un réseau de bus*, Proceedings of 18th International Conference on Association Francophone d'Interaction Homme-Machine (Montreal, Canada, 2006), pp. 241- 244.
- [Meye85]
Meyer, B., *On formalism in specifications*, IEEE Software, January 1985.
- [Meye88]
Meyer, B., *10 Tips for Getting Useful Information from Users*, IEEE Software, Volume 5, Publisher IEEE Computer Society Press, Issue 4, July 1988, pp. 89-90.
- [Mill03]
Miller, J., Mukerji, J., *MDA Guide Version 1.0.1*, Available online: <http://www.omg.org>.
- [Mont05]
Montero, F., López-Jaquero, V., Vanderdonckt, J., Gonzalez, P., Lozano, M., D., *Solving the Mapping Problem in User Interface Design by Seamless Integration in IdealXML*, Proc. of DSV-IS'2005, Springer-Verlag, Berlin, 2005.
- [Mori04]
Mori, G., Paternò, F., Santoro, C. , *Design and Development of Multidevice User Interfaces through Multiple Logical Descriptions*, IEEE Transactions on Software Engineering, August 2004, pp. 507-520.
- [Myer00]
Myers, B., A., Hudson, S., E., Pausch, R., F., *Past, present and future of user interface software tools*, ACM Trans. Computer-Human Interaction, Volume 7, No. 1, 2000, pp. 3-28.

N

- [Nava06]
Navarre, D., Palanque, P., Dragicevic, P., Bastide, R., *An approach integrating two complementary model based environments for the construction of multimodal interactive applications*, Interacting with Computers, Volume 18 (5), ISSN 0953-5438, Elsevier Science Inc. New York, NY, USA, September 2006, pp. 910-941.
- [Newm91]
Newman, S. E., Marshall, C., C., *Pushing Toulmin Too Far: Learning From an Argument Representation Scheme*, 1991, Xerox Parc Technical Report, No. SSL-92-45.
- [Niel88]
Nielsen, J., *Coordinating user interfaces for consistency*, Workshop of CHI'88, 15-16 May, 1988.
- [Niga94]
Nigay, L., *Conception et modélisation. Logicielle des Systèmes Interactif: Application aux Interface Multimodales*, Thèse de doctorat, Université Joseph Fournier, Grenoble, 1994.
- [Niga96]
Nigay, L., Coutaz, J., *Espaces conceptuels pour l'interaction multimédia et multimodale*, TSI, Multimédia et Collecticiel, Volume 15, no. 9, 1996, AFCET and Hermes Publishers, pp. 1195-1225.
- [Niga97a]
Nigay, L., Coutaz, J., *A Generic Platform for Addressing the Multimodal Challenge*, Conference on Human Factors in Computing Systems, Proceedings of the SIGCHI conference on Human

References

factors in computing systems (Denver, Colorado, United States, 1995), ISBN 0-201-84705-1, pp.: 98-105.

[Niga97b]

Nigay, L., Coutaz, J., *Multifeature Systems: The CARE Properties and Their Impact on Software Design*, 1997, Intelligence and Multimodality in Multimedia Interfaces: Research and Applications, AAAI Press Publ. CD-ROM, J. Lee Edition, 1997.

[Niga97c]

Nigay, L., Coutaz, J., *A design space for multimodal systems: Concurrent processing and Data fusion*, Proceedings of 12th-BCS conference on Human Computer Interaction, HCI'97, Springer Verlag.

[Norm86]

Norman, K., L., Weldon, L., J., Shneiderman, B., *International Journal on Man-Machine Studies*, 1986, pp. 229-248.

O

[Olsi04]

Olsina, L., Martin., M., *Ontology for Software Metrics and Indicators*, Journal of Web Engineering 2(4), 2004, pp. 262-281.

[Open07]

OpenInterface Platform - Component Developer Guide, Draft V0.1. Available online:
<http://www.openinterface.org/platform/tutorial>.

[Ovia99]

Oviatt, S., *Ten myths of multimodal interaction*, Communications of the ACM, Volume 42, Issue 11, November 1999, ISSN 0001-0782, ACM Press, New York, USA, pp.: 74-81.

[Ovia97]

Oviatt, S., DeAngeli, A., Kuhn, K., *Integration and synchronization of input modes during multimodal human-computer interaction*, Proceedings of Conference on Human Factors in Computing Systems, CHI'97 (Atlanta, GA, March 22-27, 1997), ACM Press, New York, pp. 415-422.

P

[Pala03]

Palanque, Ph., Schyn, A., *A Model-Based Approach for Engineering Multimodal Interactive*, Proceedings of 9th IFIP TC13 International Conference on Human-Computer Interaction, Interact'2003 (Zurich, 1-5 September 2003), IOS Press, Amsterdam, 2003, pp. 543-550.

[Pate97]

Paternò, F., Mancini C., Meniconi, S., *ConcurTaskTree: A diagrammatic notation for specifying task models*, Proceedings of IFIP TC 13 International Conference on Human-Computer Interaction Interact'97 (Sydney, July 14-18, 1997), Howard S., Hammond, J., Lindgaard, G. (Eds.), Kluwer Academic Publishers, Boston, 1997, pp. 362-369.

[Puer02a]

Puerta, A., Eisenstein, J., *XIML: A Common Representation for Interaction Data*, Proceedings of 6th International Conference on Intelligent User Interfaces IUP'2002 (San Francisco, USA, January 13-16, 2002), ACM Press, pp. 214-215.

[Puer02b]

Puerta, A., Eisenstein, J., *XIML: A Universal Language for User Interfaces*, Technical document, 2002.

[Plomp02]

Plomp, J., Keränen, H., Nikkola, H., Y, Rantakokko, T., *Supporting past, present and future interaction with home appliances*, International ITEA Workshop on Virtual Home Environments (February 20-21, 2002), Paderborn, Germany.

Q

[QVT05]

Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification, Final Adopted Specification, November, 2005.

References

R

- [Rous05] Rousseau, C., Bellik, Y., Vernier, F., *Multimodal output specification/ simulation platform*, Proceedings of 7th International Conference on Multimodal Interfaces ICMF'2005 (Trento, 4-6 October 2005), ACM Press, New York, 2005, pp. 84-91.

S

- [Scha07] Schaffer, R., *A Survey on Transformation Tools for Model Based User Interface Development*, Proceedings of HCI'International 2007, J. Jacko (Ed.): Human-Computer Interaction, Part I, LNCS 4550, Springer, Berlin, pp. 1178–1187.
- [Scha06] Schaefer, R., Steffen, B., Wolfgang, M., *Task Models and Diagrams for User Interface Design*, Proceedings of 5th International Workshop, TAMODIA'2006 (Hasselt, Belgium, October 2006), Lecture Notes in Computer Science, Vol. 4385, Springer Verlag Berlin, 2006, pp. 39-53.
- [Schu92] Schumacher, R., *Phone-based interfaces: reaserch and guidelines*, Ameritech Services, Inc., Proceedings of the Human Factor Society, 36th annual meeting, 1992, pp. 1051-1055.
- [Schü97] Schürr, A., *Programmed Graph Replacement Systems*, Handbook of Graph Grammars and Computing by Graph Transformation, Rozenberg G. (Ed.), Volume 1: Foundations, World Scientific, Singapore, 1997, pp. 479-546.
- [Schy05] Schyn, A, *Une approche fondée sur les modèles pour l'ingénierie des systèmes interactif multimodaux*, Thèse de doctorat, Université Toulouse III, 2005.
- [Shne86] Shneiderman, B., Shafer, P., Simon, R., Weldon, L., J., *Display strategies for program browsing: concepts and experiment*, IEEE Software, 3, 1986, pp. 7-15.
- [Shne98] Shneiderman, B., *Designing the user interface: strategies for effective human-computer interaction*, 3rd Edition, Publisher Addison-Wesley, ISBN 0-201-69497-2, March 1998.
- [Shne06] Shneiderman, B., Fischer, G., *Creativity Support Tools: Report From a U.S. National Science Foundation Sponsored Workshop*, International Journal of Human-Computer Interaction, 20(2), pp. 61–77.
- [Sieg88] Siegel, S., Castellan, N., J., *Nonparametric Statistics for The Behavioral Sciences*, McGraw-Hill, Inc., second edition, 1988.
- [Sinh01] Sinha A., K., Klemmer S., R., Chen, J., Landay J., A., Chen C., *SUEDE: Iterative, Informal Prototyping for Speech Interfaces*, Video poster in Extended Abstracts of Human Factors in Computing Systems CHI 2001 (Seattle, WA, March 31-April 5, 2001), pp. 203-204.
- [Sott07] Sottet, J.-S., Calvary, G., Coutaz, J., Favre, J.-M., Vanderdonckt, J., Stanciulescu, A., Lepreux, S., *A Language Perspective on the Development of Plastic Multimodal User Interfaces*, Journal of Multimodal User Interfaces, Vol. 1, No. 2, June 2007, pp. 1-12.
- [Stan04] Stanciulescu, A., Limbourg, Q., Vanderdonckt, J., *Graficul – modalitate de reprezentare a elementelor interfeței cu utilizatorul*, Proceedings of 1st National Conference on Computer-Human Interaction ROCHI'2004 (Bucharest, September 23-24, 2004), Ștefan Trăușan-Matu, S., Pribeanu, C. (Eds.), Polytechnic University of Bucharest, Bucharest, 2004.
- [Stan05] Stanciulescu, A., Limbourg, Q., Vanderdonckt, J., Michotte, B., Montero, F., *A Transformational Approach for Multimodal Web User Interfaces based on UsiXML*, Proceedings of 7th International Conference on Multimodal Interfaces ICMF'2005 (Trento, 4-6 October, 2005), ACM Press, New York, 2005, pp. 259-266.
- [Stan06] Stanciulescu, A., Vanderdonckt, J., *Design Options for Multimodal Web Applications*, Proceedings of 6th International Conference on Computer-Aided

References

- Design of User Interfaces CADUI'2006 (Bucharest, Romania, 6-8 June 2006), Chapter 4, Springer-Verlag, Berlin, 2006, pp. 41-56.
- [Stan07] Stanculescu, A., Vanderdonckt, J., Macq, B., *Automatic Usability Assessment of Multimodal User Interfaces Based on Ergonomic Rules*, Proceedings of E-Mode Joint Workshop on Multimodal Interfaces 2007 (Paris, 27-28 September 2007), S. Praud (ed.).
- [Stan08] Stanculescu, A., Vanderdonckt, J., Mens, T., *Colored Graph Transformation Rules for Model-Driven Engineering of Multi-Target Systems*, Proceedings of 3rd International Workshop on Graph and Model Transformation GraMoT'2008 (Leipzig, May 12, 2008), ACM Press, New York, 2008, pp.37-44.
- [Suhm99] Suhm, B., Myres, B., Waibel, A., *Model-based and empirical evaluation of multimodal interactive error correction*, Proceedings of the SIGCHI conference on Human Factors in Computing Systems (Pittsburgh, Pennsylvania, United States, 1999), pp. 584 – 591.
- [Sun07] Sun, Y., Shi, Y., Chen, F., Chung, V., *An efficient unification-based multimodal language processor in multimodal input fusion*, Proceedings of 2007 Conference of Computer-Human Interaction, Special Interest Group (CHISIG) of Australia on Computer-Human Interaction (Adelaide, Australia, 2007), pp. 215 – 218.
- T**
- [Tour02] Touraine, D., Bourdot, P., Bellik, Y., Bolot, L., *A framework to manage multimodal fusion of events for advanced interactions within Virtual Environments*, Proceedings of 8th Eurographics Workshop on Virtual Environments, Eurographics Association Publ., Barcelona, 30-31 May 2002, pp. 159-168.
- U**
- [USIX05] UsiXML Consortium, *UsiXML, a General Purpose XML Compliant User Interface Description Language*, UsiXML V1.6.3, 16 June 2005. Available online: <http://www.usixml.org>.
- [USIX07] UsiXML Consortium, *UsiXML, a General Purpose XML Compliant User Interface Description Language*, UsiXML V1.8, 14 February 2007. Available online: <http://www.usixml.org>.
- V**
- [Vand01] Vanderdonckt, J., Bouillon, L., Souchon, N., *Flexible Reverse Engineering of Web Pages with Vaquita*, Proceedings of WCRE'2001, IEEE 8th Working Conference on Reverse Engineering, Stuttgart, October, 2001, IEEE Press.
- [Vand03] Vanderdonckt, J., Limbourg, Q., Florins, M., *Deriving the Navigational Structure of a User Interface*, Proceedings of 9th IFIP TC 13 International Conference on Human-Computer Interaction INTERACT'2003 (Zurich, 1-5 September 2003).
- [Vand07] Vanderdonckt, J., Coutaz, J., Calvary, G., Stanculescu, A., *Multimodality for Plastic User Interfaces: Models, Methods, and Principles*, Multimodal User Interfaces: from signals to interaction, D. Tzovaras (ed.), Chap. 3, Lecture Notes in Electrical Engineering, Springer-Verlag, Berlin, 2007, pp. 79-105.
- [Vand08] Vanderdonckt, J., Calvary, G., Coutaz, J., *Multimodality for Plastic User Interfaces: Models, Methods and Principles*, D. Tzovaras (Ed.) Multimodal User Interfaces. Signals and Communication Technology, Springer-Verlag Berlin Heidelberg 2008, DOI: 10.1007/978-3-540-78345-9, pp.75-101.
- [Varr02] Varró, D., Varró, G., Pataricza, A., *Designing the Automatic Transformation of Visual Languages*, Science of Computer Programming, 44, 2002, pp. 205–227.

References

W

[Winc08]

Winckler, M., Vanderdonckt, J., Stanciulescu, A., Trindade, F., *Cascading Dialog Modeling with UsiXML*, Proceedings of the Design, Specification and Verification of Interactive Systems DSV-IS'2008 (Ontario, Canada, July 16-18, 2008), T.C.N. Graham and P. Palanque (Eds.), LNCS 5136, Springer-Verlag Berlin Heidelberg 2008, pp. 121–135.

[W3C04a]

W3C consortium, *EMMLA: Extensible MultiModal Annotation markup language*, W3C Working Draft, 14 December 2004. Available online: <http://www.w3.org/TR/emma>.

[W3C04b]

W3C consortium, *Voice Extensible Markup Language (VoiceXML) Version 2.0*, W3C recommendation 16 March 2004. Available online: <http://www.w3.org/TR/voicexml20>.

[W3C04c]

W3C consortium, *XHTML+Voice Profile 1.2*, 16 March 2004. Available online: <http://www.voicexml.org/specs/multimodal/x+v/12>.

[W3C01]

W3C consortium, *XML Schema Specification*, W3C Recommendation, 2 May 2001. Available online: <http://www.w3.org/XML/schema.html>.

Appendix A. UsiXML expanded Task Model

A UsiXML Task Model is a hierarchical task structure, where each task is described by:

- An *identifier* and a *name*.
- A *category*, which is determined by the allocation of the task: a task performed by the user (e.g. a cognitive task) is called a *user task*. A task completely executed by the system (e.g. a computation task) has category *application task*. A task performed by the user in interaction with the system (e.g. viewing results, selecting items, editing a field, pushing a button to invoke an application function) is called an *interaction task*. Last, *abstraction tasks* (e.g. booking a flight) are complex tasks whose performance can not be univocally allocated and that can be decomposed into simpler tasks (thus, there must be at least two different task categories among the tasks decomposing an abstraction task).
- Optional attributes such as the task *importance* or *frequency*.

Tasks are linked by two types of relationships:

- *Decomposition relationships*. Each task can be decomposed into two or more subtasks. Thus, with the exception of the root task, each task has a mother task from which the temporal relationships are inherited.
- *Temporal relationships*. Temporal relationships between the tasks are specified with *temporal operators*. The temporal operators are based upon the LOTOS operators.

Temporal relationships are of two types: unary and binary. Unary operators characterize a single task when binary operators link together two sibling tasks.

There are three unary operators. The first one is the iteration operator (notation: T*), which means that the task T is repeated until some other task disables it. The second one, is the finite iteration operator (notation T(n)), used when the designer knows in advance exactly how many time the task will be performed. The last operator permits indicating that the performance of a task is optional (notation [T]).

If we consider two generic tasks T1 and T2, the binary temporal operators can be described as follows:

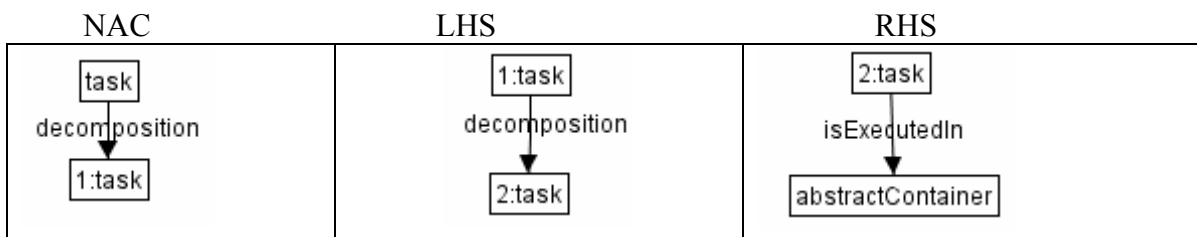
1. *Independent concurrency* or *parallelism* (T1 ||| T2): T1 and T2 can be performed in any order without any constraints. E.g.: filling field 1 and field 2 in a form.
2. *Concurrency* (or parallelism) *with information exchange* (T1 ||| T2): T1 and T2 can be performed in any order but they have to synchronize in order to exchange information. E.g. filling field 1 and field 2 in a form when there is some coherency check (between a phone number and a city for example).

Appendix A UsiXML expanded Task Model

3. *Deterministic choice* ($T1 \parallel T2$): Once one task is initiated, the other cannot be accomplished anymore, until the first task is terminated. E.g. log in as a reviewer or as an author on a conference reviewing system.
4. *Non-deterministic choice* ($T1 \pi T2$): Once one task is finished the other cannot be accomplished anymore. E.g., saving one's bank statements to one's desktop computer or printing them in the bank's self-service lobby.
5. *Order independency* or *sequential independence* ($T1 \mid = \mid T2$). This operator is equivalent to ($T1 \gg T2$) OR ($T2 \gg T1$) E.g., in a hospital, the human task of taking blood samples from patients can be done before or after filling the request form for lab analysis, but both tasks have to be completed before the request is send to the lab.
6. *Disabling* ($T1 \mid > T2$): T1 is definitively disabled when T2 (or the first subtask or T2) has been performed. E.g., sending a form disables all tasks that could be achieved in this form.
7. *Suspend-resume* ($T1 \mid > T2$): T1 is interrupted when T2 (or its first subtask) is performed. Once T2 terminated, T1 is reactivated from the state reached before the interruption. E.g., an alarm message indicating that the battery of the device is low interrupts any activity on that device, and the activity is reactivated only when the alarm dialog box is closed.
8. *Enabling* ($T1 \gg T2$): T2 is enabled when T1 is terminated. For instance, the authentication of the user allows him/her to access to the restricted area of a web site.
9. *Enabling with information passing* ($T1 \parallel \gg T2$): T1 enables T2 and provides it some information. E.g., T1 allows the user to specify a query and T2 displays the search results related to the information requested in T1.

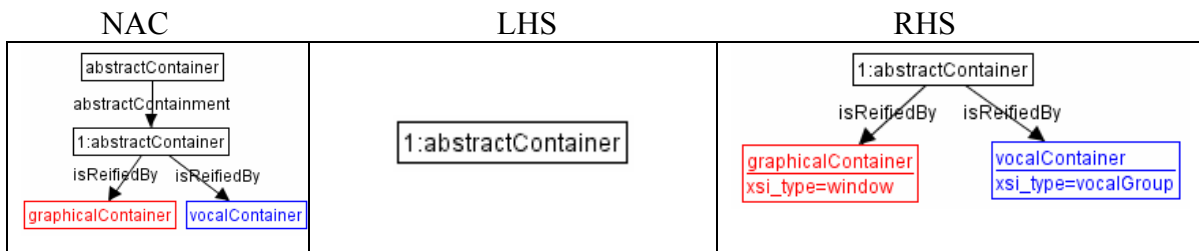
Appendix B. Transformation rule catalog

- I. Transformation rules that support the design options
 - 1. Transformation rules for sub-task triggering: not supported
 - 2. Transformation rules for sub-task presentation:
 - 2.1. Separated



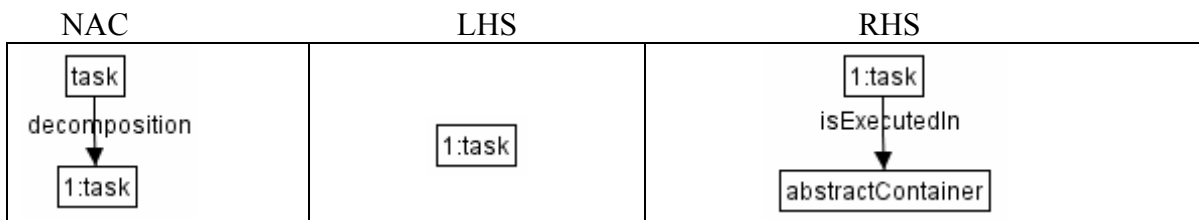
Rule 1. Generate abstract containers for each sub-task of the same task

Concretization for MM UI:

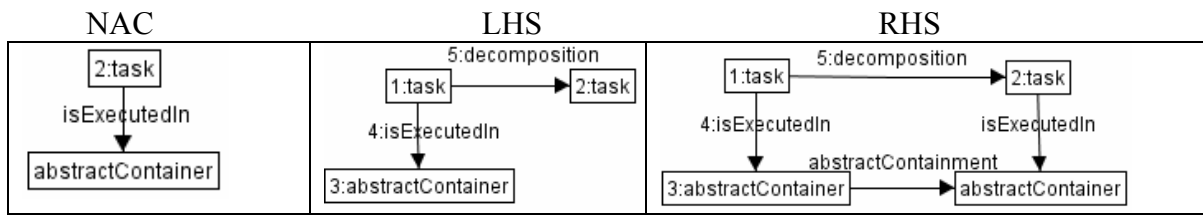


Rule 2. Generation of a window and a vocalGroup for each top-level AC

2.2. Combined

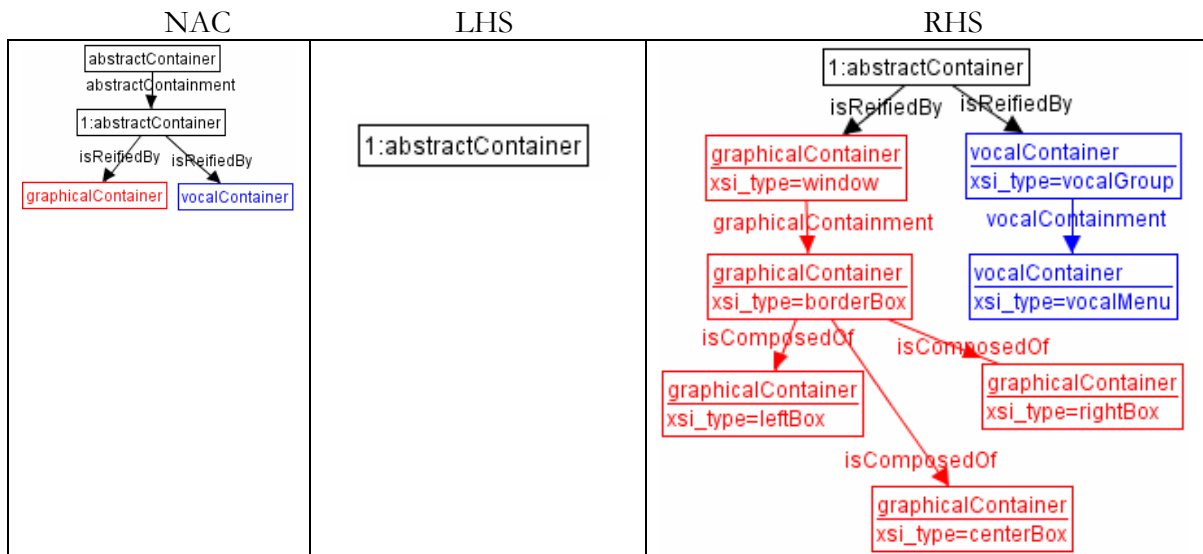


Rule 3. Generate an abstract container for the father task

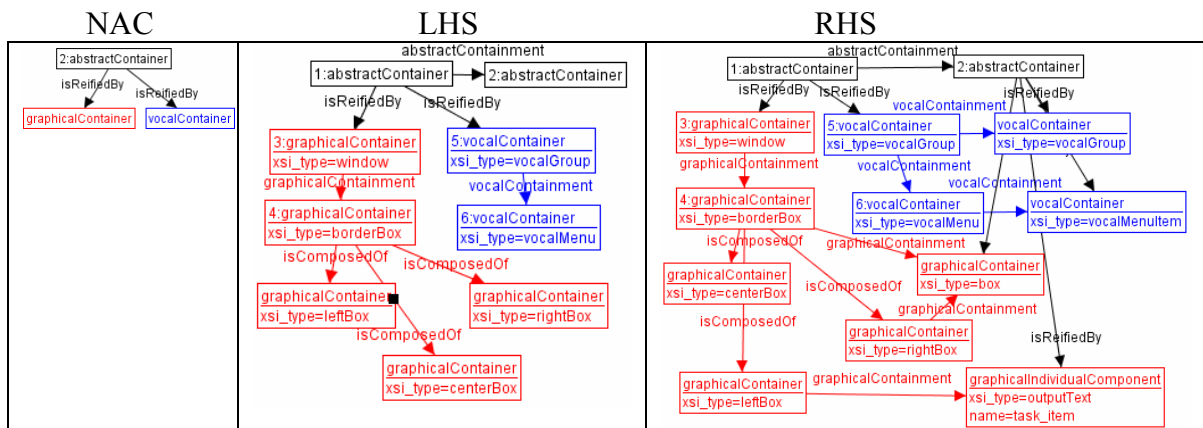


Rule 4. Generate an abstract container for each sub-task of the father task

- One at once
 - ✓ Extended task list:
- Concretization for MM UI:



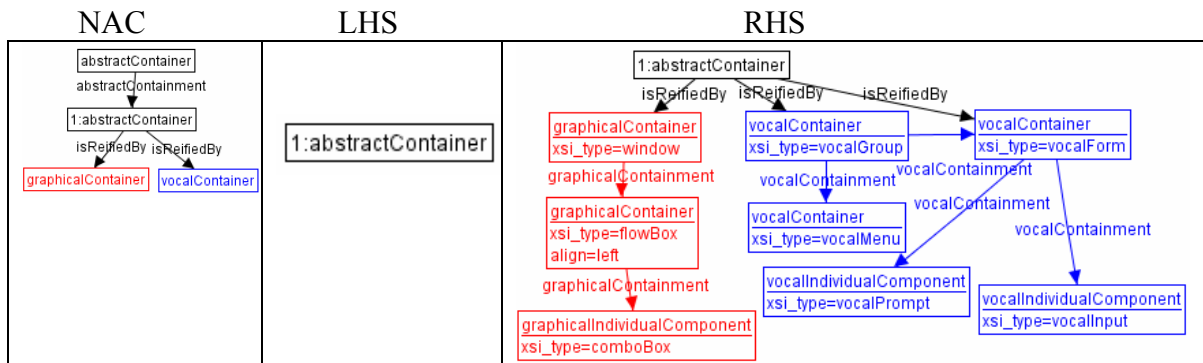
Rule 5. Reification of the top most AC into a window containing a borderBox (with left, center and right boxes) and a vocalGroup containing a vocalMenu



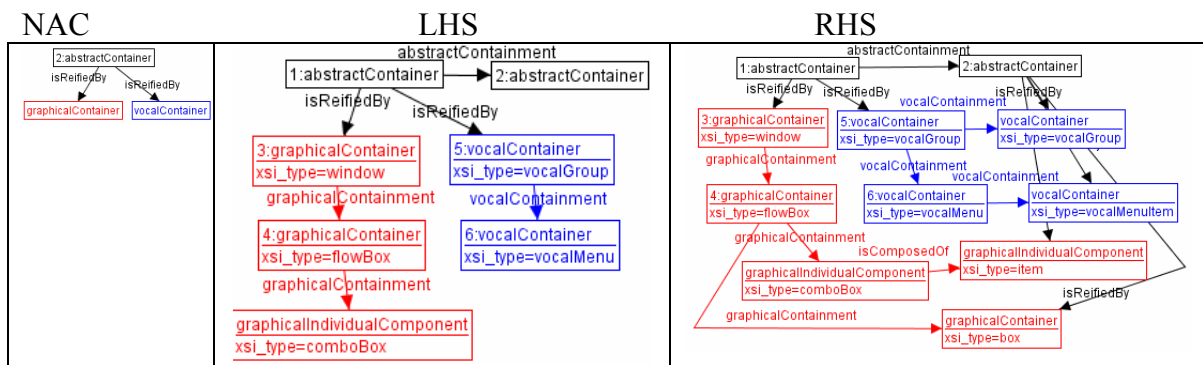
Rule 6. Generation of a box, an outputText (task item) a vocalGroup and a vocalMenuItem for each AC embedded into the top most AC

Appendix B Transformation rule catalog

- ✓ **Reduced task list:**
Concretization for MM UI:

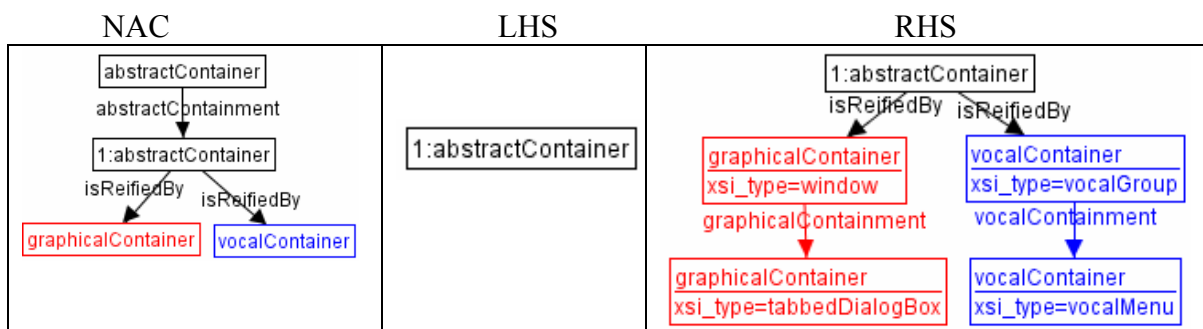


Rule 7. Reification of the top most AC into a window containing a flowBox with a comboBox and a vocalGroup containing a vocalMenu and a vocalForm with vocalPrompt and vocalInput



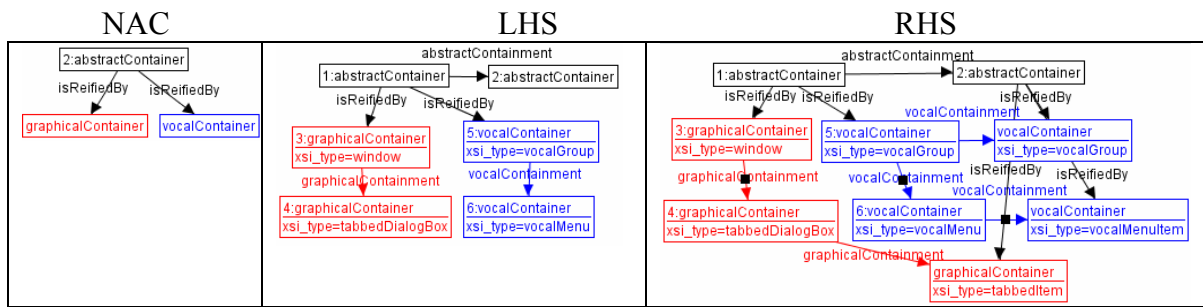
Rule 8. Generation of a box, a comboBox item, a vocalGroup and a vocalMenuItem for each AC embedded into the top most AC

- ✓ **Tabbed list:**
Concretization for MMUIs:



Rule 9. Reification of the top most AC into a window containing a tabbedDialogBox and a vocalGroup containing a vocalMenu

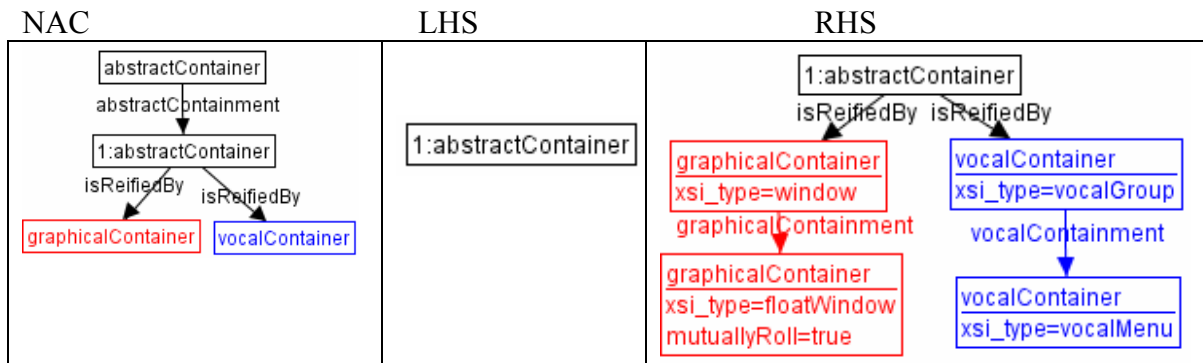
Appendix B Transformation rule catalog



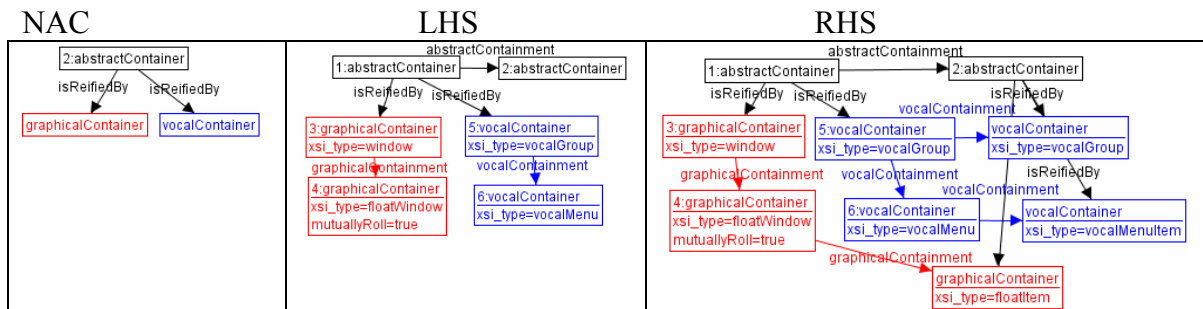
Rule 10. Generation of a tabbedItem, a vocalGroup and a vocalMenuItem for each AC embedded into the top most AC

✓ Single expansion list

Concretization for MM UI:

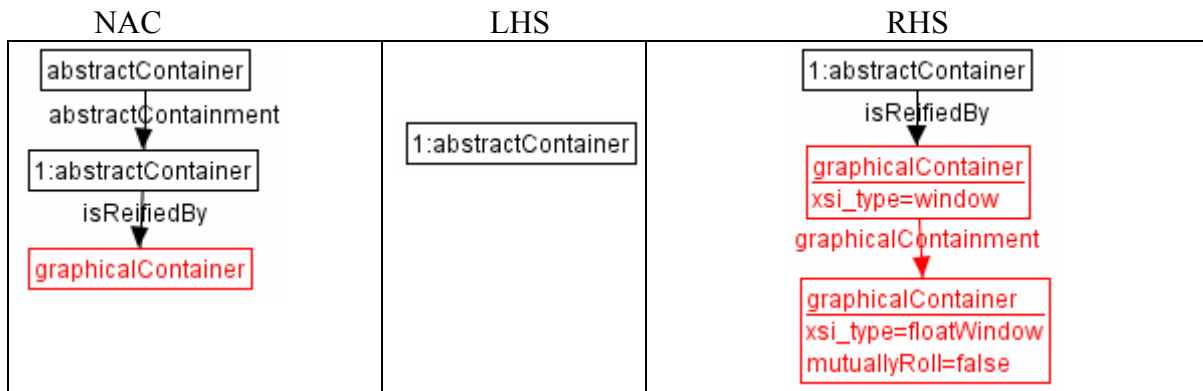


Rule 11. Reification of the top most AC into a window containing a floatWindow and a vocalGroup containing a vocalMenu

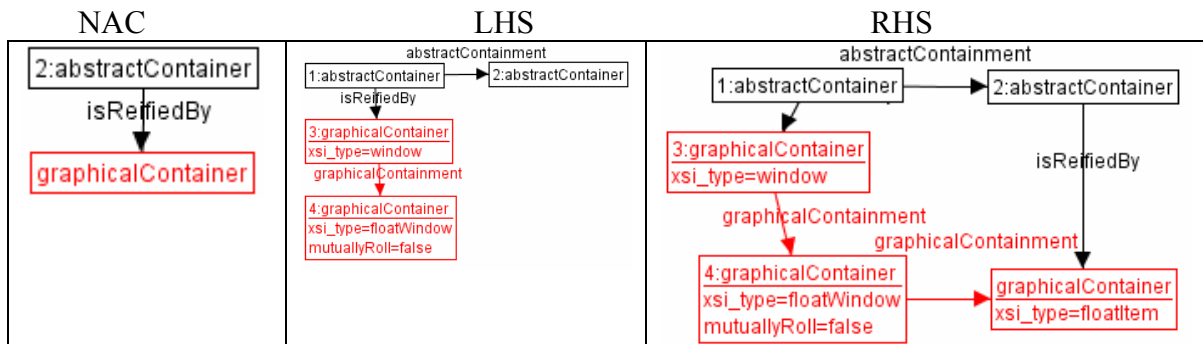


Rule 12. Generation of a floatItem, a vocalGroup and a vocalMenuItem for each AC embedded into the top most AC

- Many at once
 - ✓ Multiple expansion list:
- Concretization for GUI:

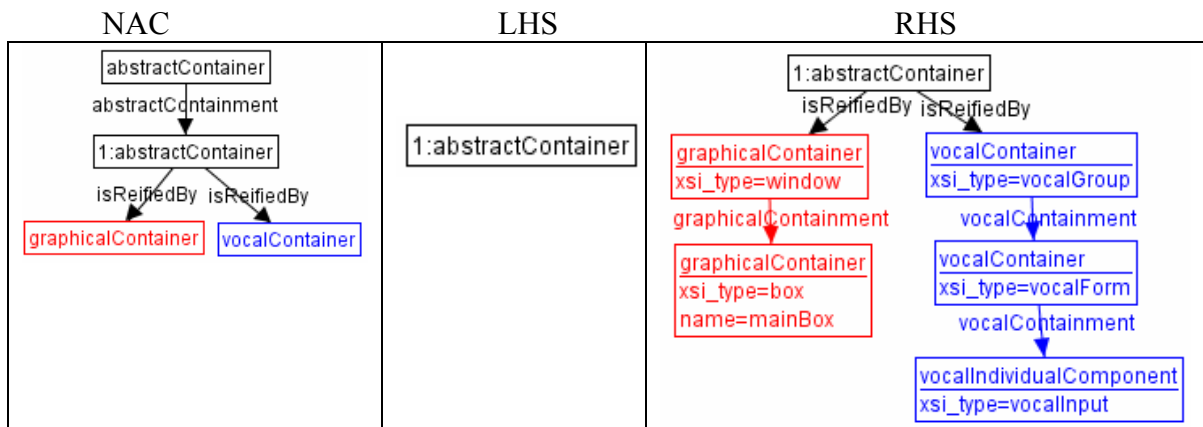


Rule 13. Reification of the top most AC into a window containing a floatWindow



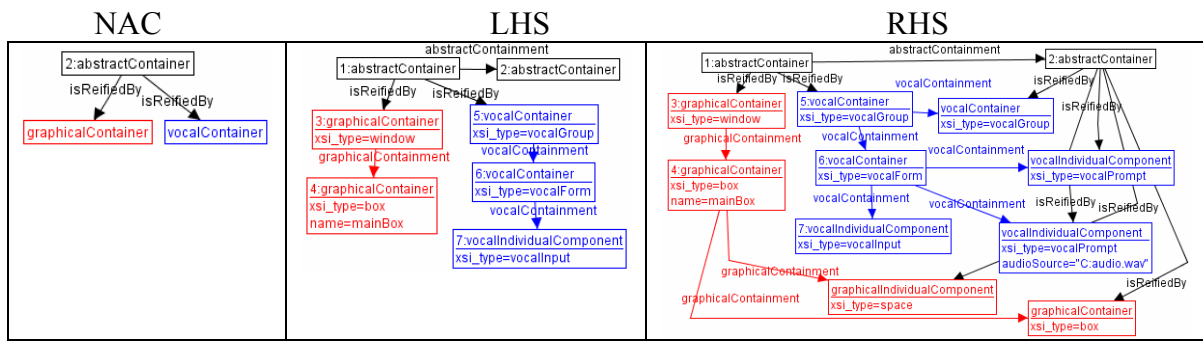
Rule 14. Generation of a floatItem for each AC embedded into the top most AC

- All at once
 - ✓ Separated list:
- Concretization for MM UI:



Rule 15. Reification of the top most AC into a window containing a box and a vocalGroup containing a vocalForm with a vocalInput

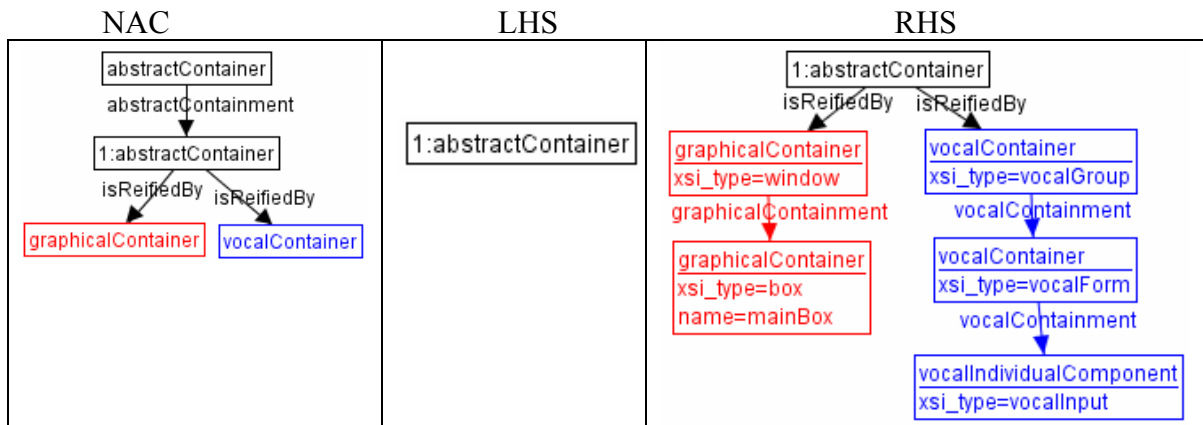
Appendix B Transformation rule catalog



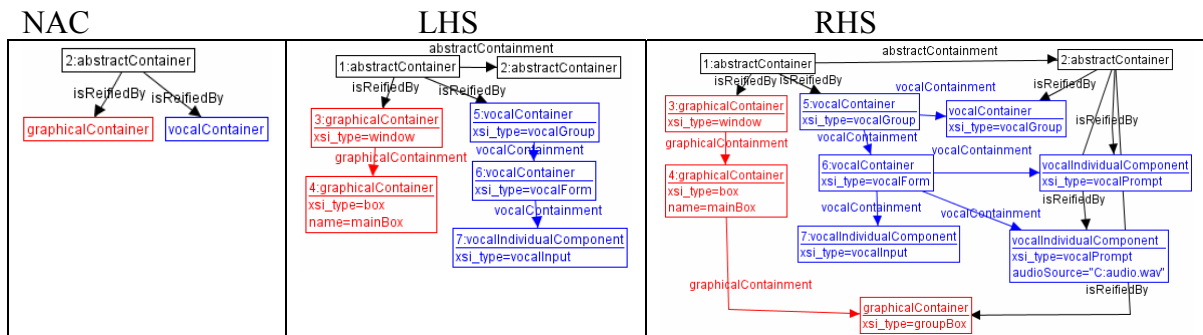
Rule 16. Generation of a box, a space, a vocalGroup and two vocalPrompts for each AC embedded into the top most AC

✓ Grouped list:

Concretization for MM UI:



Rule 17. Reification of the top most AC into a window containing a box and a vocalGroup containing a vocalForm with a vocalInput

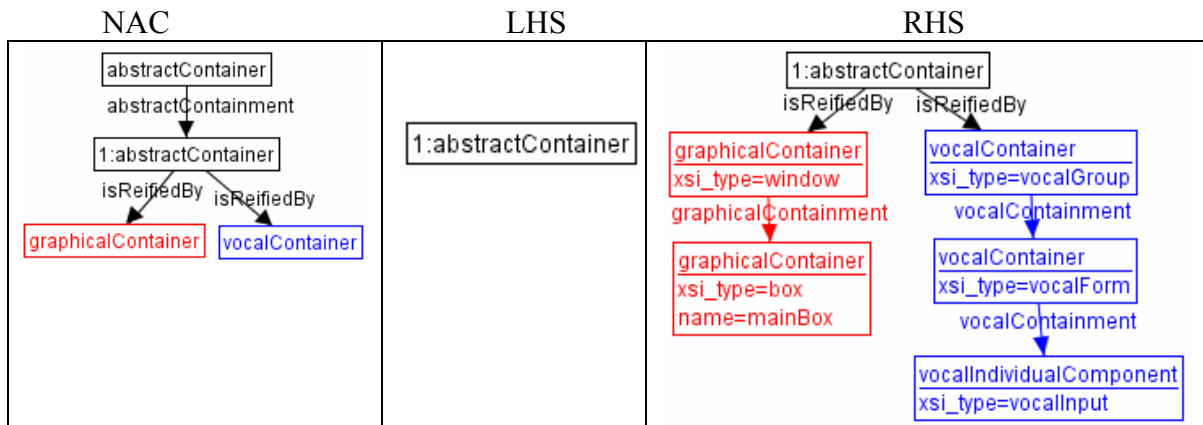


Rule 18. Generation of a groupBox, a vocalGroup and two vocalPrompts for each AC embedded into the top most AC

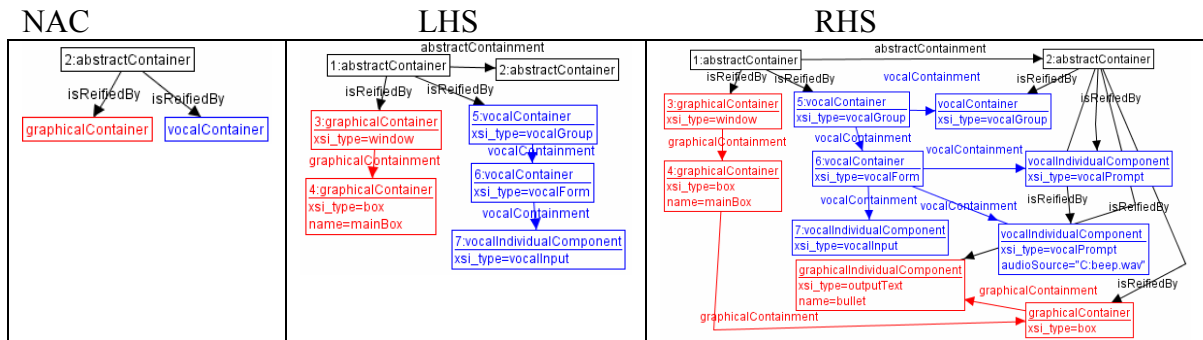
Appendix B Transformation rule catalog

✓ **Bulleted list:**

Concretization for MMUIs:



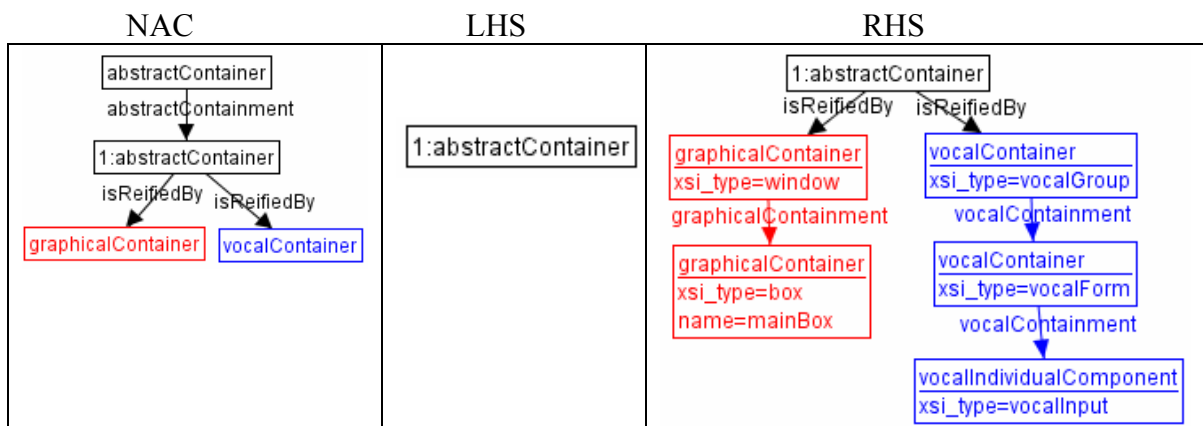
Rule 19. Reification of the top most AC into a window containing a box and a vocalGroup containing a vocalForm with a vocalInput



Rule 20. Generation of a box that contains an outputText (bullet), a vocalGroup and two vocalPrompts for each AC embedded into the top most AC

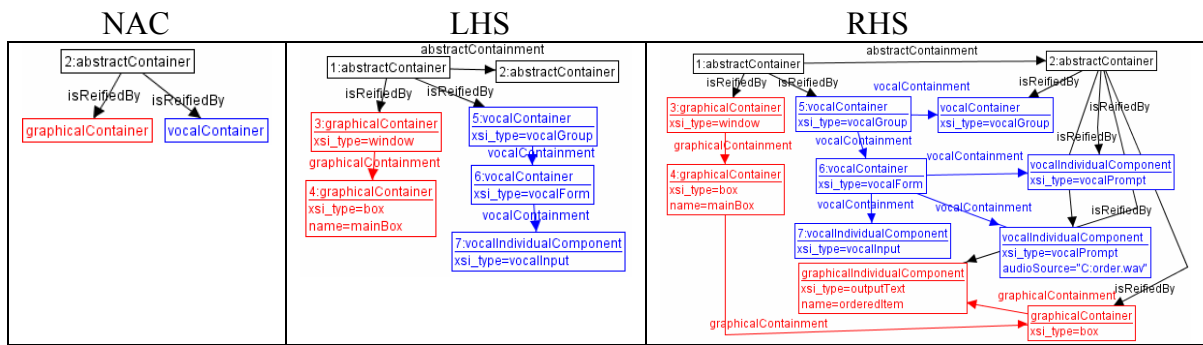
✓ **Ordered list:**

Concretization for MM UI:



Rule 21. Reification of the top most AC into a window containing a box and a vocalGroup containing a vocalForm with a vocalInput

Appendix B Transformation rule catalog

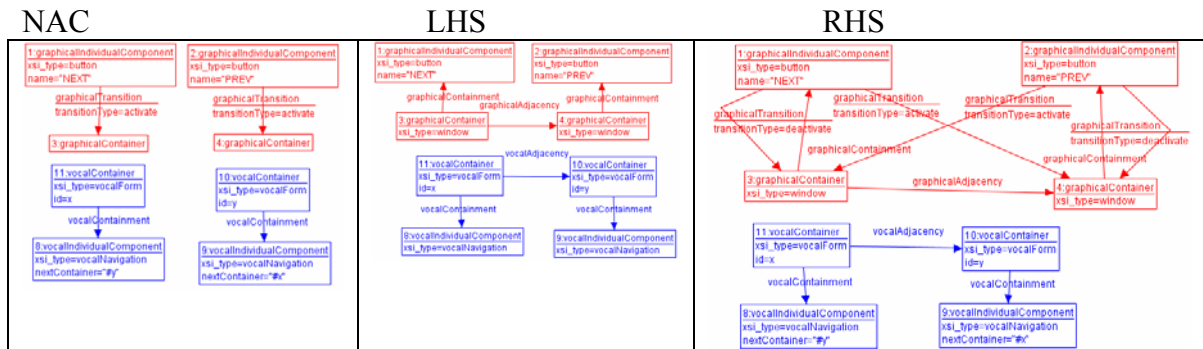


Rule 22. Generation of a box that contains an outputText (orderedItem), a vocalGroup and two vocalPrompts for each AC embedded into the to most AC

3. Transformation rules for sub-task navigation:

✓ Sequential

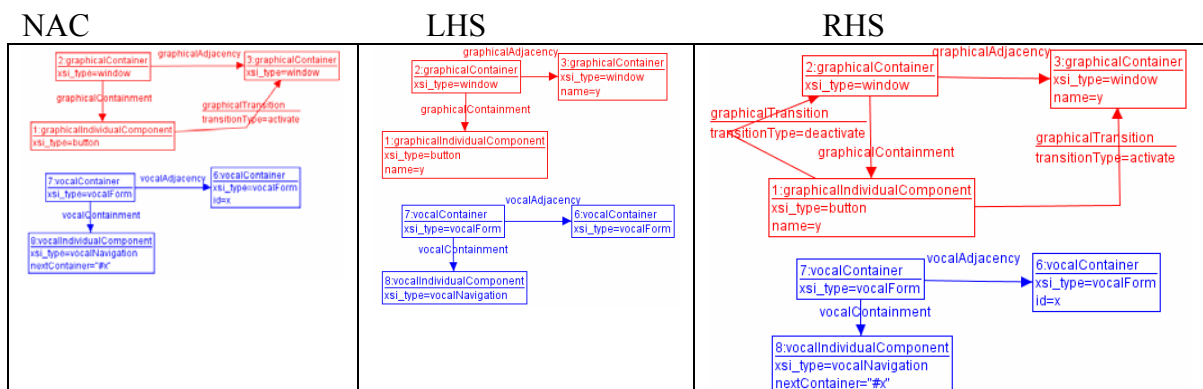
Concretization for MM UI:



Rule 23. Generation of a graphicalTransition relationships that endow the (PREV, NEXT) buttons with activation and deactivation features over adjacent and current GCs, respectively and vocalNavigation specification to ensure the navigation between sequential vocalForms

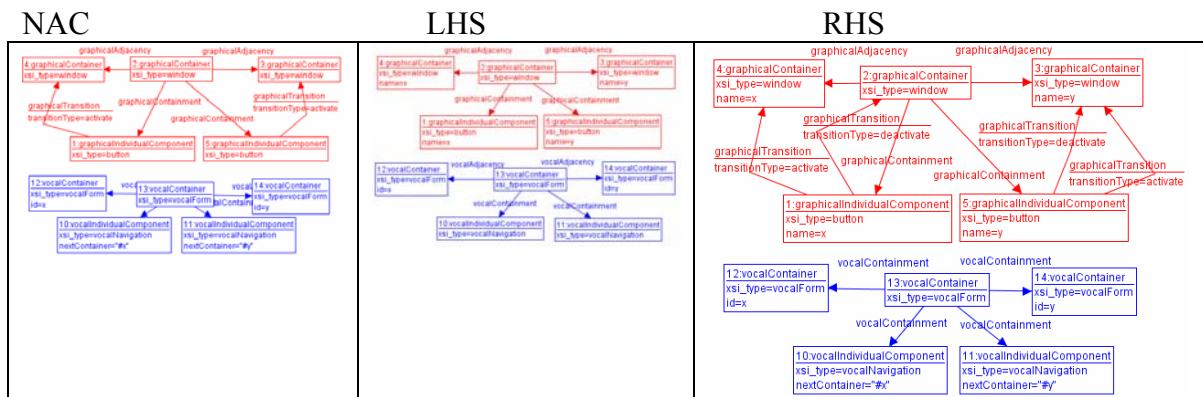
✓ Asynchronous

Concretization for MM UI:

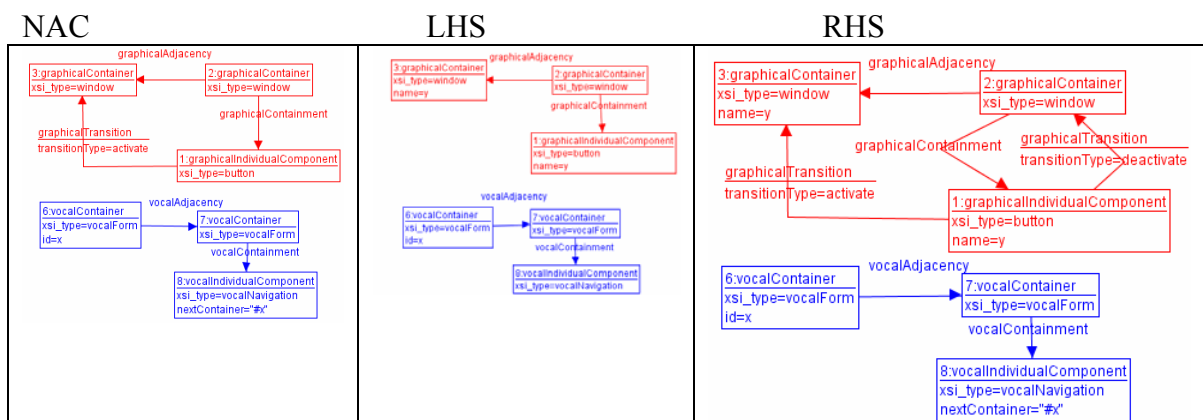


Rule 24. Generation of graphicalTransition relationships for buttons placed in the first GC that ensure the forward navigation and vocalNavigation placed in the first vocalForm that ensure the navigation towards the second vocalForm

Appendix B Transformation rule catalog



Rule 25. Generation of graphicalTransition relationships for buttons placed in the middle GCs and vocalNavigation specification in order to ensure the forward and backward navigation and

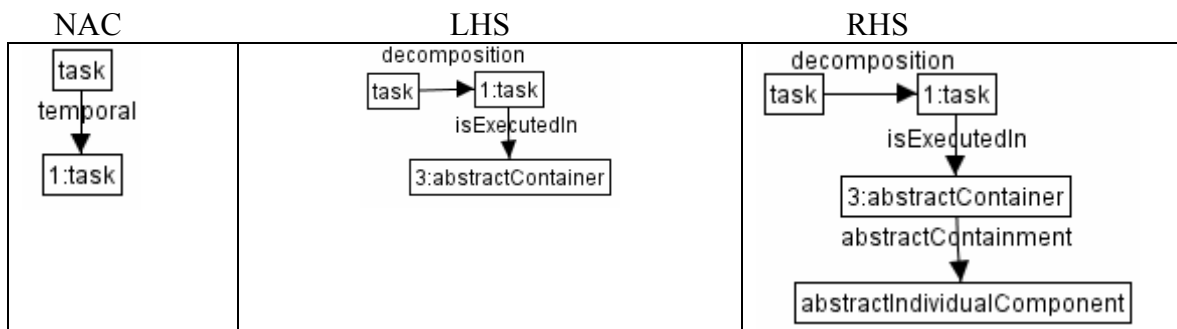


Rule 26. Generation of graphicalTransition relationships for buttons placed in the last GC and vocalNavigation specification in order to ensure the backward navigation

4. Transformation rules for navigation type:

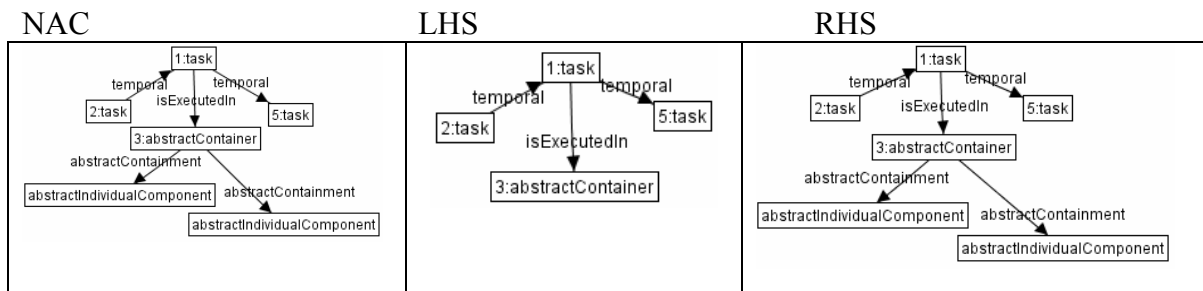
4.1. Containment:

4.1.1. Local:

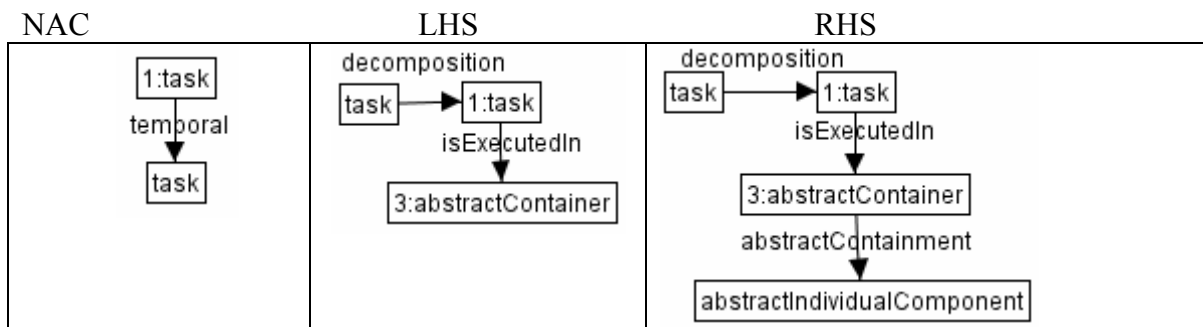


Rule 27. Generation of one local placed AIC that ensures the navigation between the first AC and the second one in any type of sub-task presentation (separated or combined)

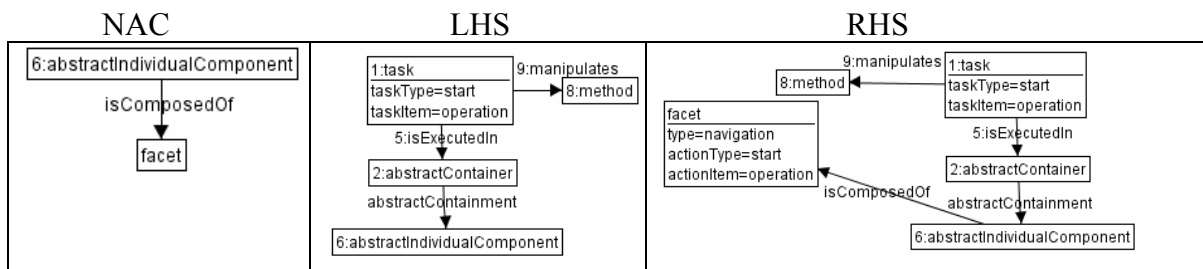
Appendix B Transformation rule catalog



Rule 28. Generation of two local placed AICs that ensure the navigation for middle placed ACs in any type of sub-task presentation (separated or combined)

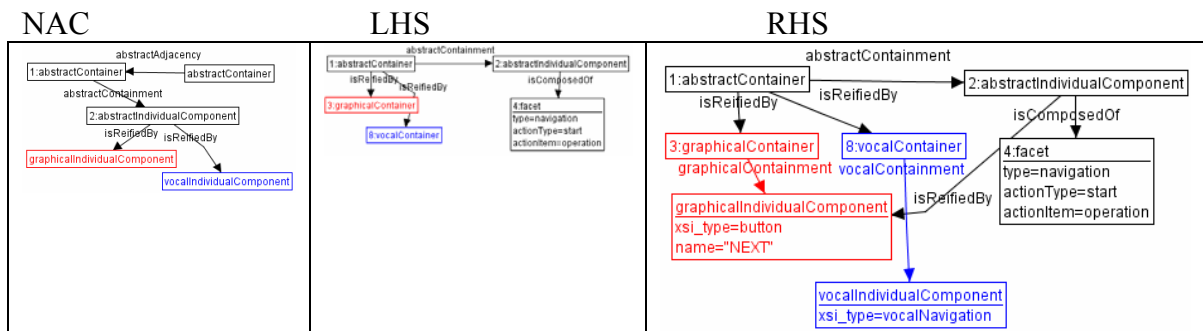


Rule 29. Generation of one local placed AIC that ensures the navigation between the last AC and the previous one in any type of sub-task presentation (separated or combined)



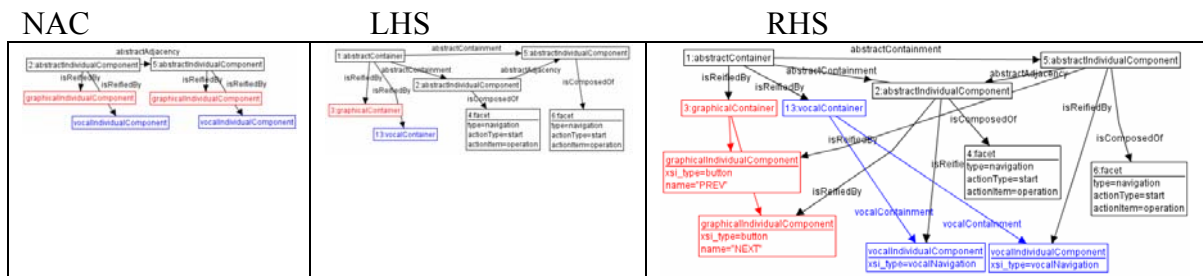
Rule 30. Creation of navigation facets for local placed AICs

Concretization for MM UI:

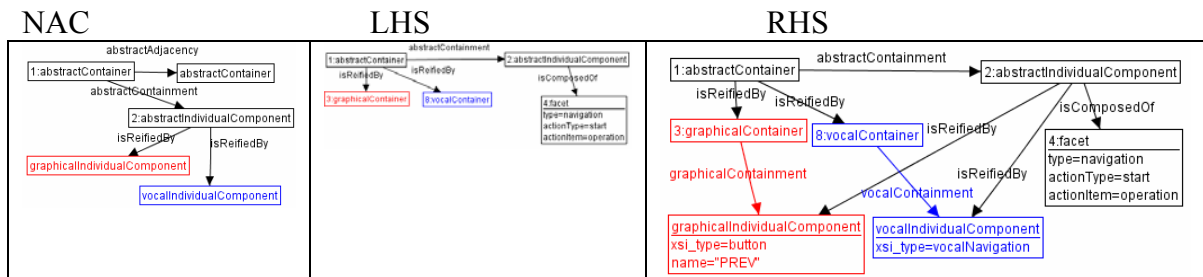


Rule 31. Generation of “NEXT” button that ensures the navigation between the first GC and the second GC and vocalNavigation element that ensures the navigation between the first VC and the second one in any type of sub-task presentation (separated or combined)

Appendix B Transformation rule catalog

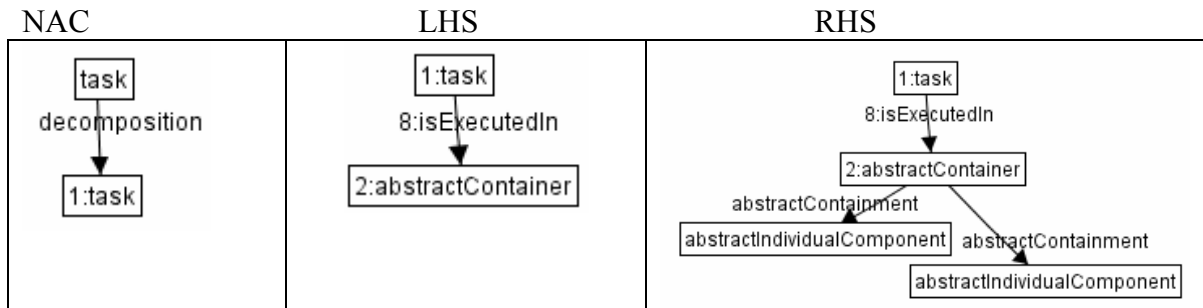


Rule 32. Generation of “PREV” and “NEXT” buttons that enure the navigation for middle placed GCs and vocalNavigation elements in any type of sub-task presentation (seperated or combined)

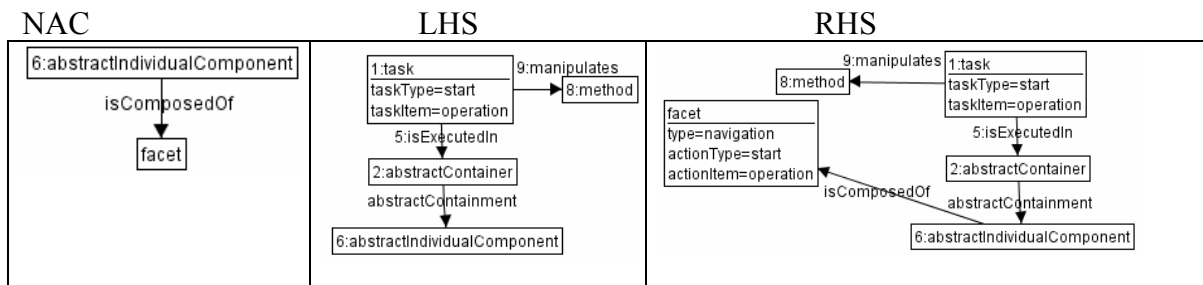


Rule 33. Generation of “PREV” button that ensures the navigation between the last GC and the previous one and vocalNavigation element that ensures the navigation between the last VC and the previous one in any type of sub-task presentation (seperated or combined)

4.1.2. Global:

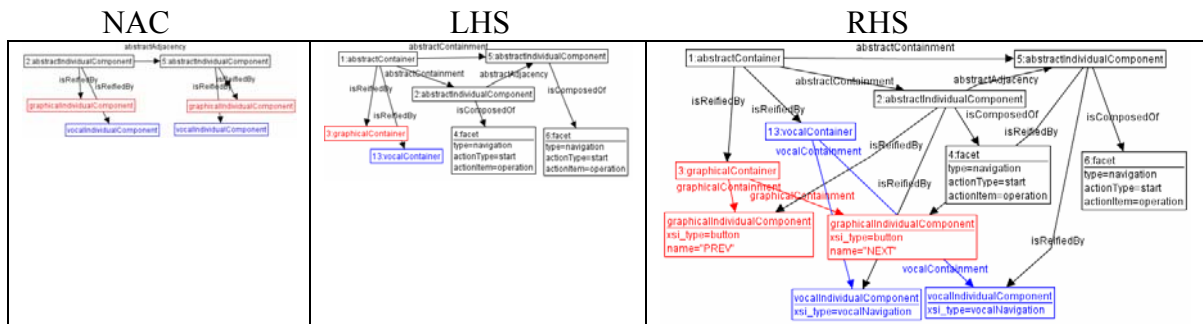


Rule 34. Generation of two global placed AICs that ensure the navigation between the sub-tasks



Rule 35. Creation of navigation facets for global placed AIC

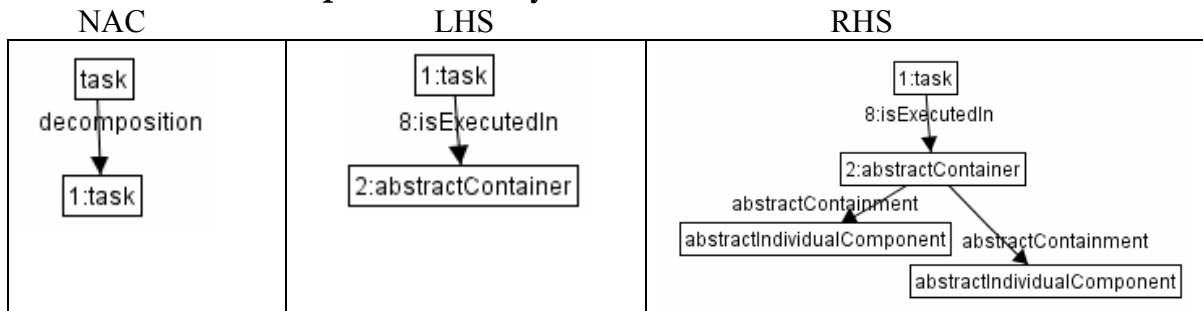
Concretization for MM UI:



Rule 36. Generation of “PREV” and “NEXT” buttons and vocalNavigation elements that ensure the global navigation

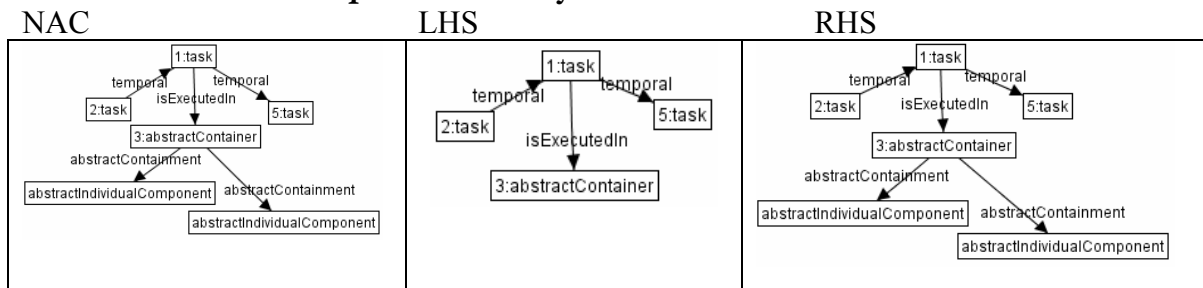
4.2. Cardinality

4.2.1. Simple cardinality:

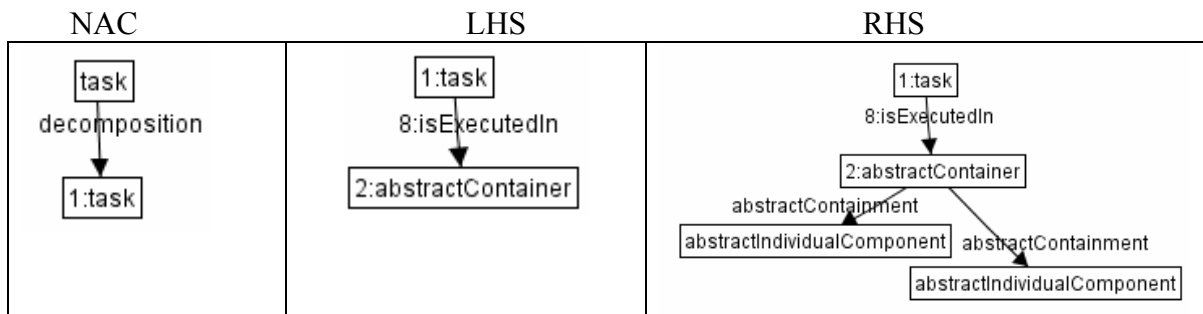


Rule 37. Generation of two AICs that will be concretized in two logically connected buttons (PREV, NEXT) ensuring the navigation between the sub-tasks

4.2.2. Multiple cardinality:



Rule 38. Generation of two local placed AICs that will be concretized in two logically connected buttons (PREV, NEXT) ensuring the navigation between sub-tasks

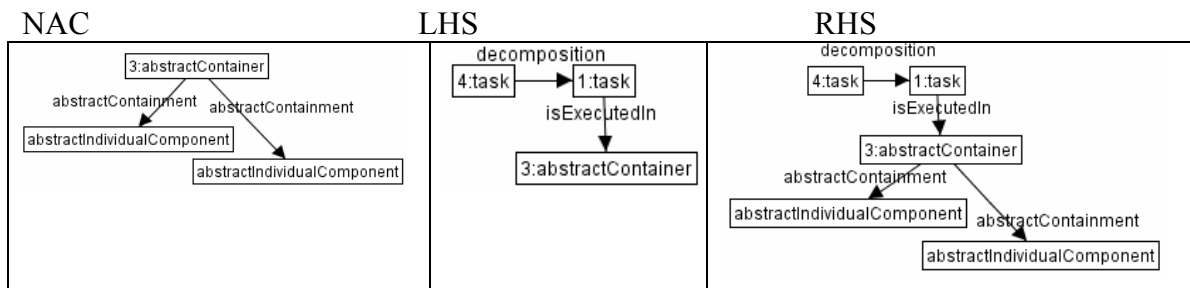


Rule 39. Generation of two global placed AICs that will be concretized in two logically connected buttons (PREV, NEXT) ensuring the navigation between the sub-tasks

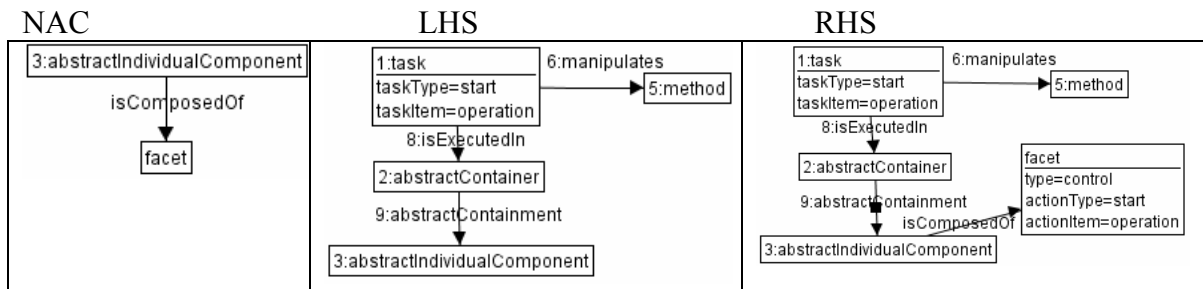
5. Transformation rules for control type:

5.1. Containment:

5.1.1. Local:

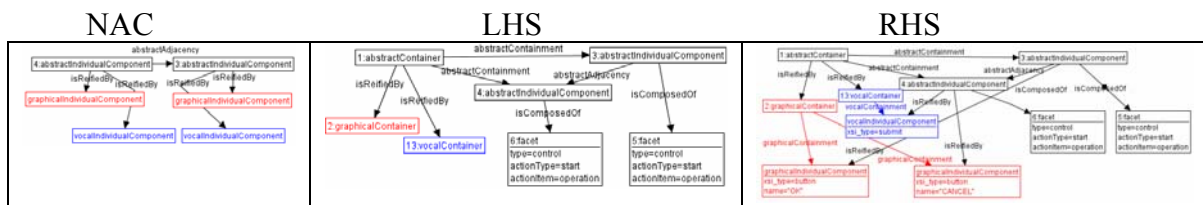


Rule 40. Generation of two local placed AICs that ensure the control of data in any type of sub-task presentation (separated or combined)



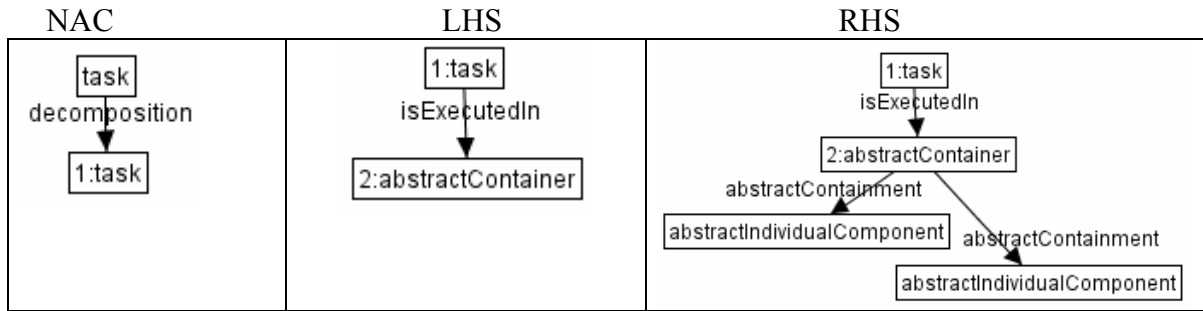
Rule 41. Generate control facet for local placed AICs

Concretization for MM UI:

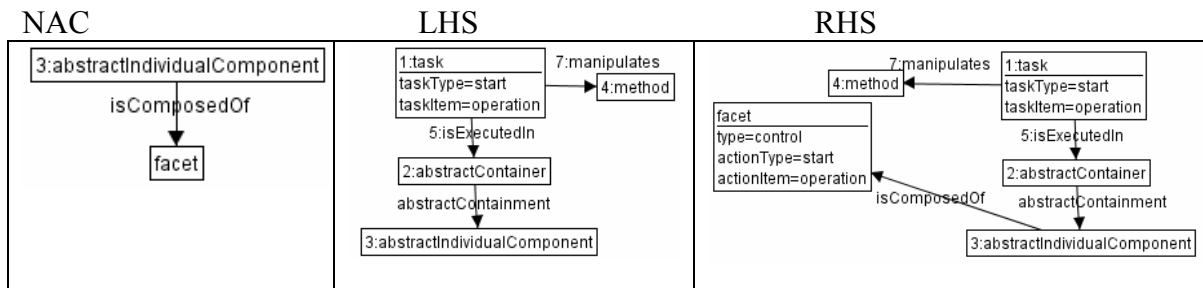


Rule 42. Generation of “OK” and “CANCEL” buttons and submit element that ensure the local control of data

5.1.2. Global:

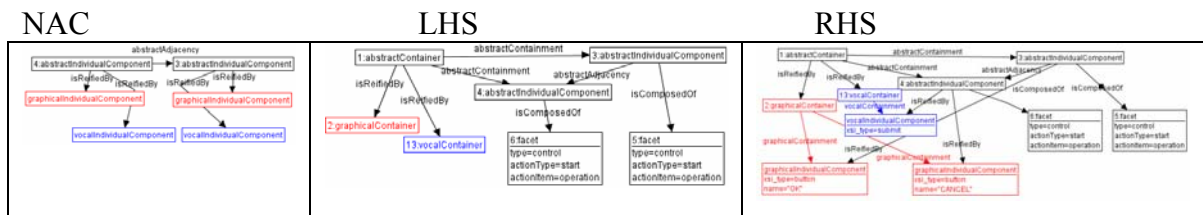


Rule 43. Generation of two global AICs that ensures the control of data



Rule 44. Creation of control facet for global placed AIC

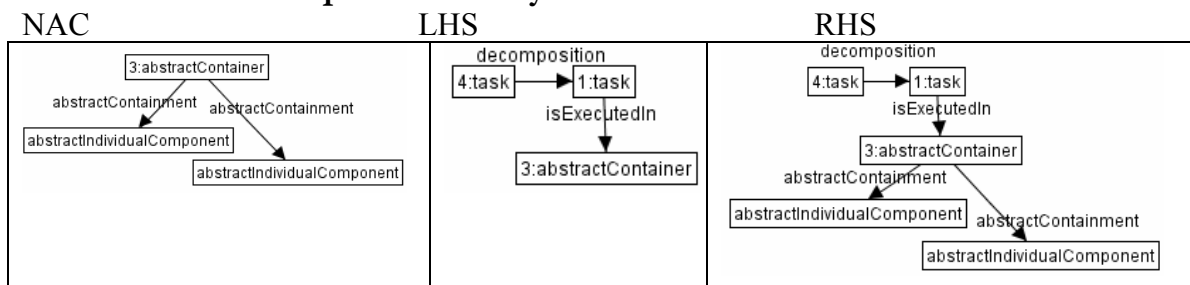
Concretization for MM UI:



Rule 45. Generation of “OK” and “CANCEL” buttons and submit element that ensure the global control of data

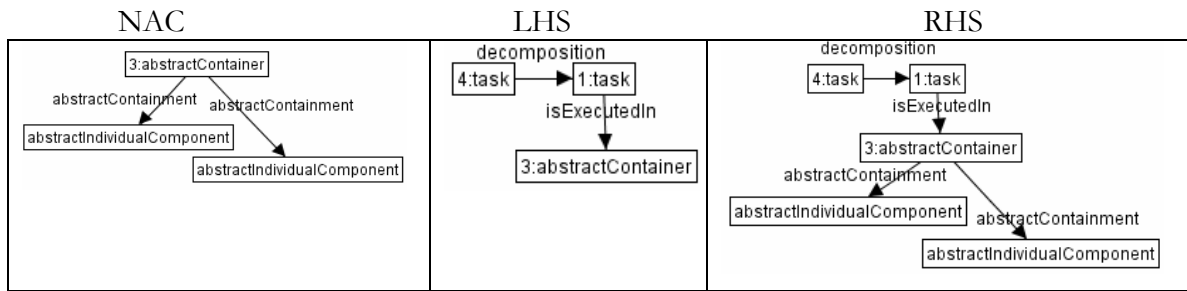
5.2. Cardinality

5.2.1. Simple cardinality:

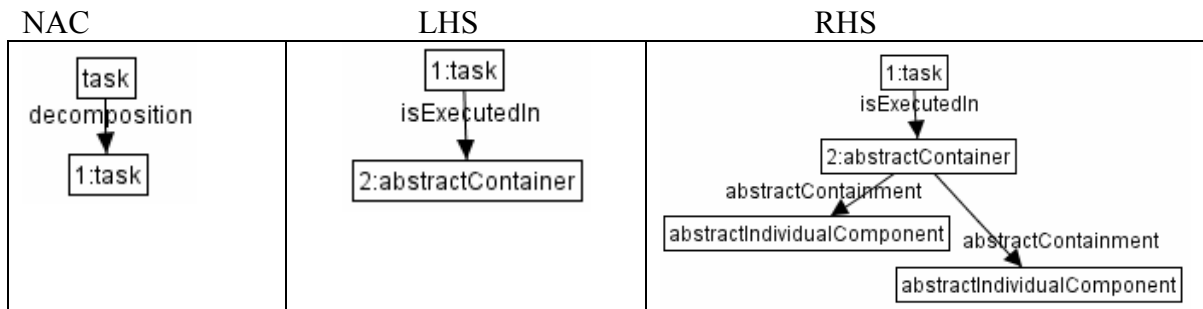


Rule 46. Generation of two AICs that will be concretized in two logically connected buttons (OK, CANCEL) ensuring the contro of data for each sub-task

5.2.2. Multiple cardinality:



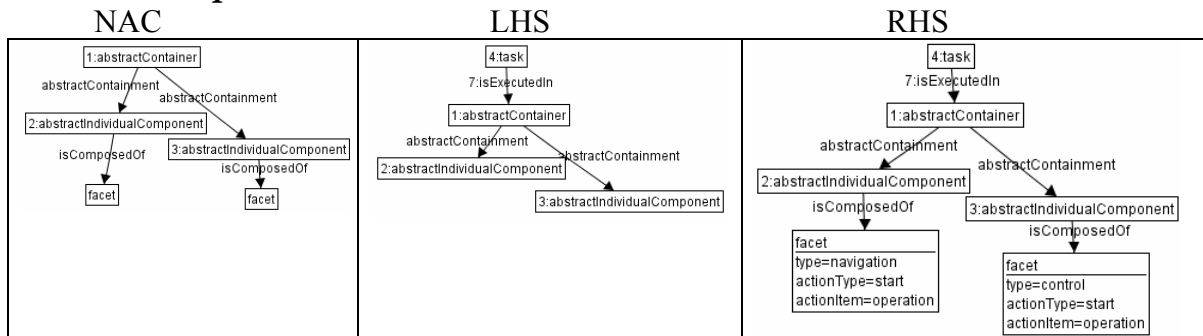
Rule 47. Generation of two AICs that will be concretized in two logically connected buttons (OK, CANCEL) ensuring the control of data for each sub-task



Rule 48. Generation of two global AICs that will be concretized in two logically connected buttons (OK, CANCEL) ensuring the control of data for the root task

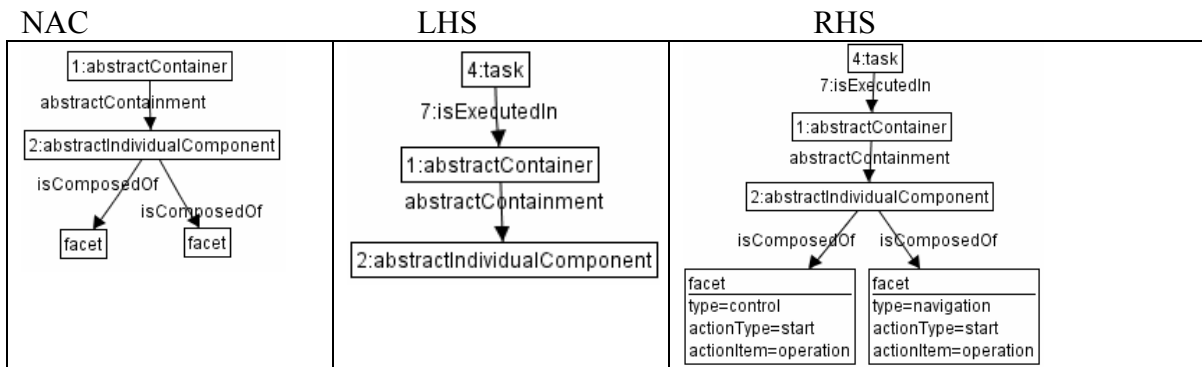
6. Transformation rules for navigation and control type:

6.1. Separated:



Rule 49. Generate separated navigation and control facets for AICs

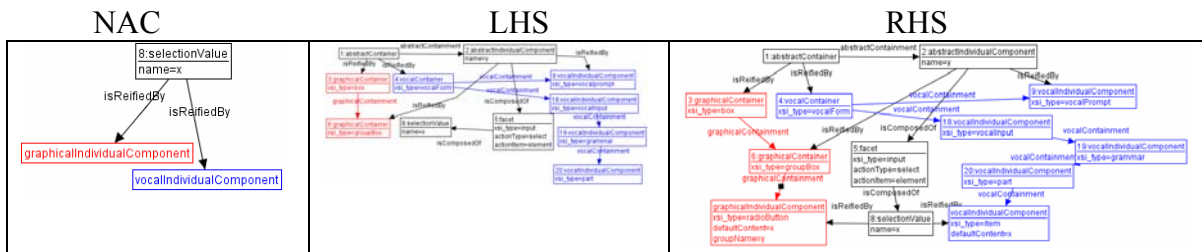
6.2. Combined:



Rule 50. Generate combined navigation and control facets for AICs

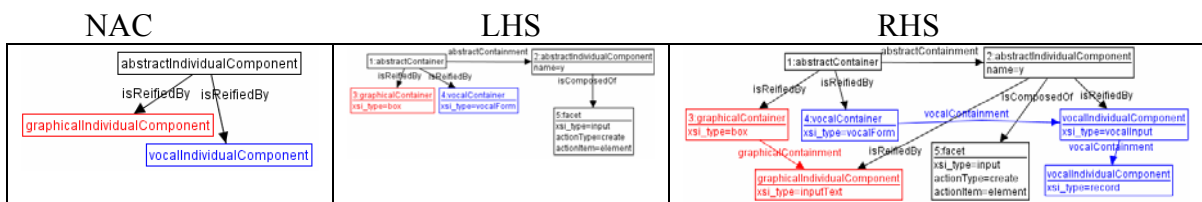
7. Transformation rules for sub-task guidance:

7.1. Guided Concretization for MM UI:



Rule 51. Generate radioButtons and vocal items of a grammar that will guide the user with the possible options

7.2. Unguided Concretization for MM UI:



Rule 52. Generates inputText and vocalInput elements that do not guide the user with the possible options

8. Transformation rules for support for default value and unit: not supported

9. Transformation rules for answer cardinality:

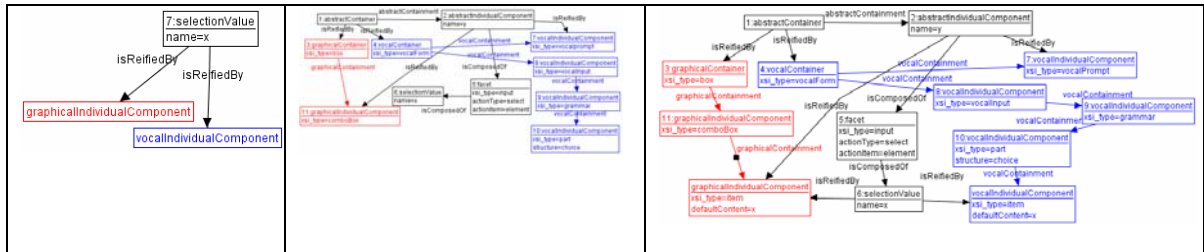
9.1. Simple

Concretization for MM UI:

NAC

LHS

RHS



Rule 53. Generation of comboBox items and grammar items that enable single selection among multiple options

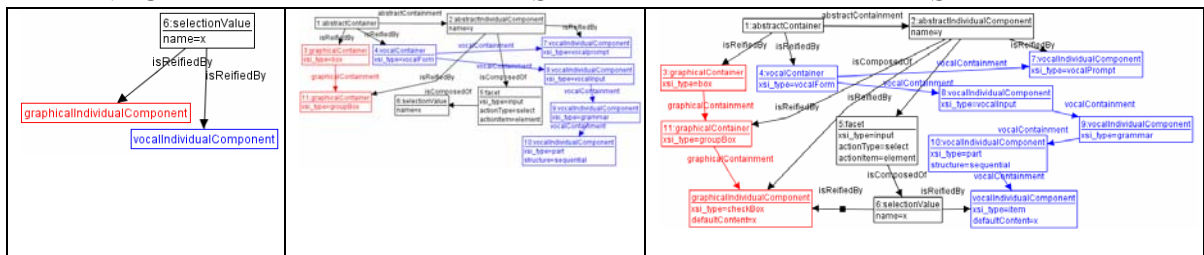
9.2. Multiple

Concretization for MM UI:

NAC

LHS

RHS



Rule 54. Generation of checkBox items and grammar items that enable single selection among multiple options

10. Transformation rules for confirmation answer:

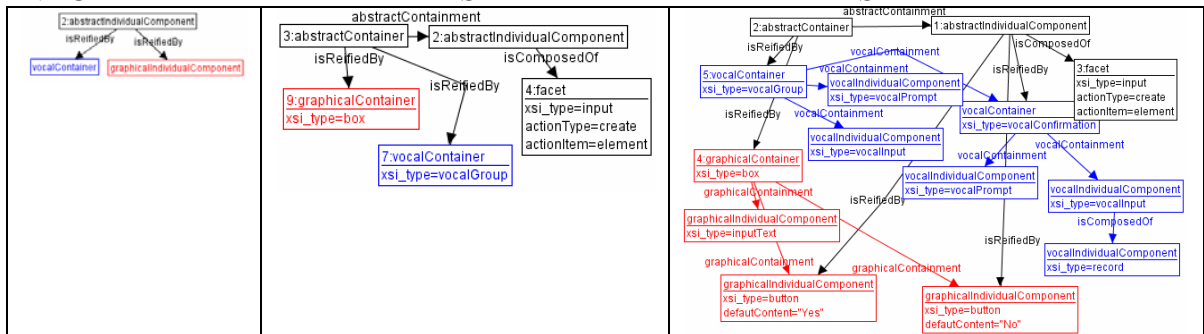
10.1. With confirmation:

Concretization for MM UI:

NAC

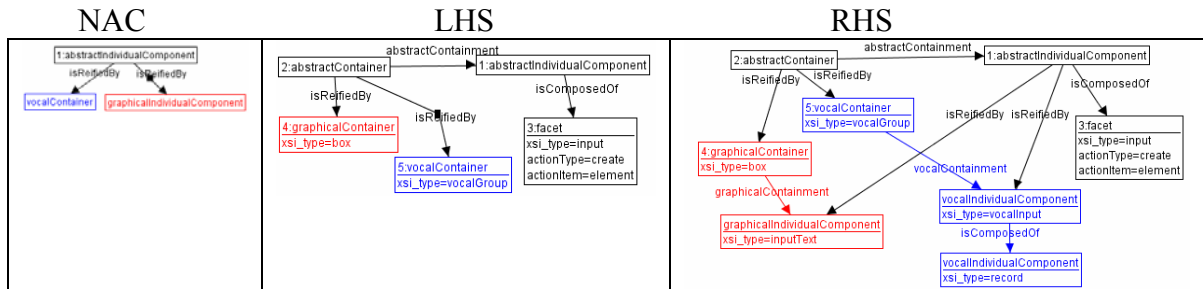
LHS

RHS



Rule 55. Generation of inputText and vocalInput that require confirmation

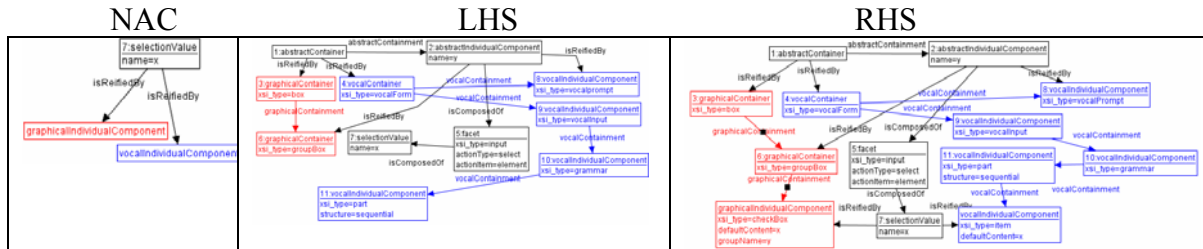
10.2. Without confirmation
 Concretization for MM UI:



Rule 56. Generation of inputText and vocalInput that do not require confirmation

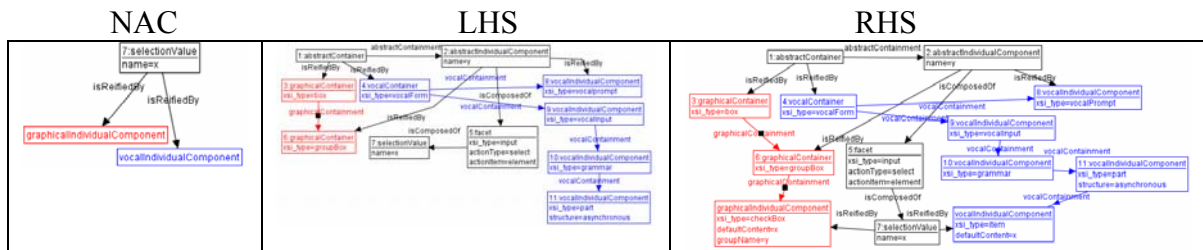
11. Transformation rules for answer order:

11.1. Order dependent
 Concretization for MM UI:



Rule 57. Generation of grammar items that require sequential uttering

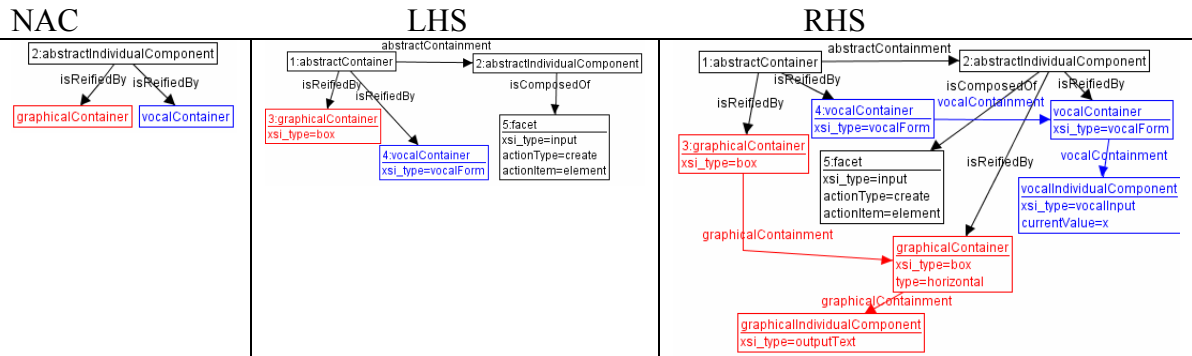
11.2. Order independent
 Concretization for MM UI:



Rule 58. Generation of grammar items that require asynchronous uttering

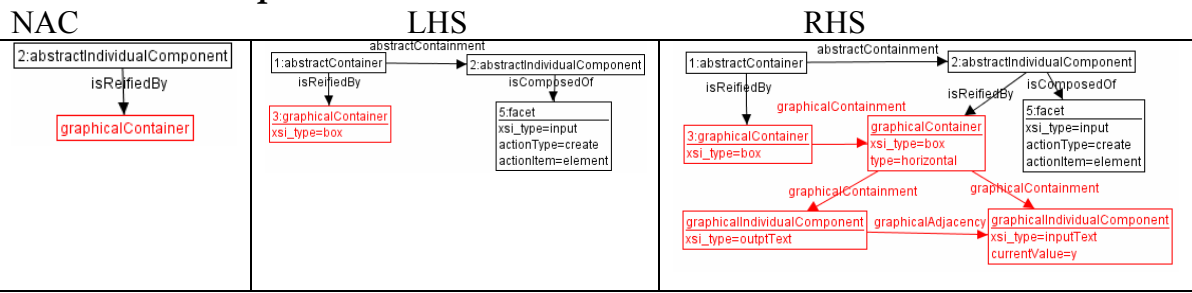
12. Transformation rules for input

12.1. Vocal



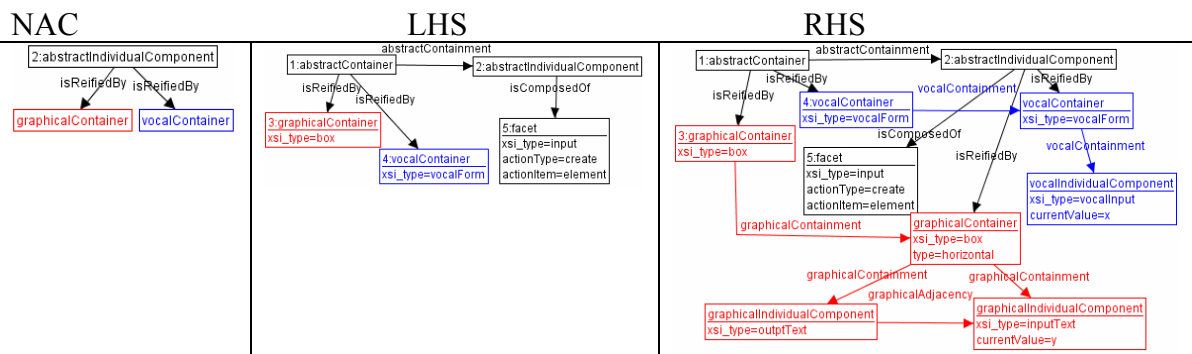
Rule 59. Generation of vocalInput components

12.2. Graphical



Rule 60. Generation of inputText components

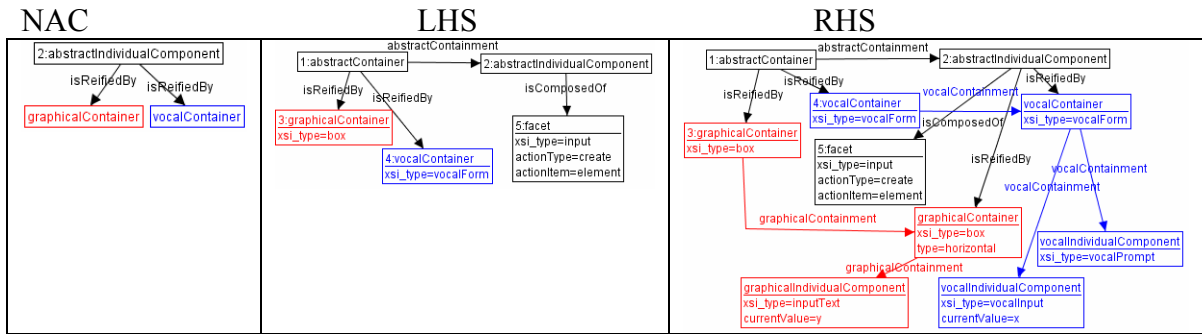
12.3. Multimodal



Rule 61. Generation of vocalInput and inputText components

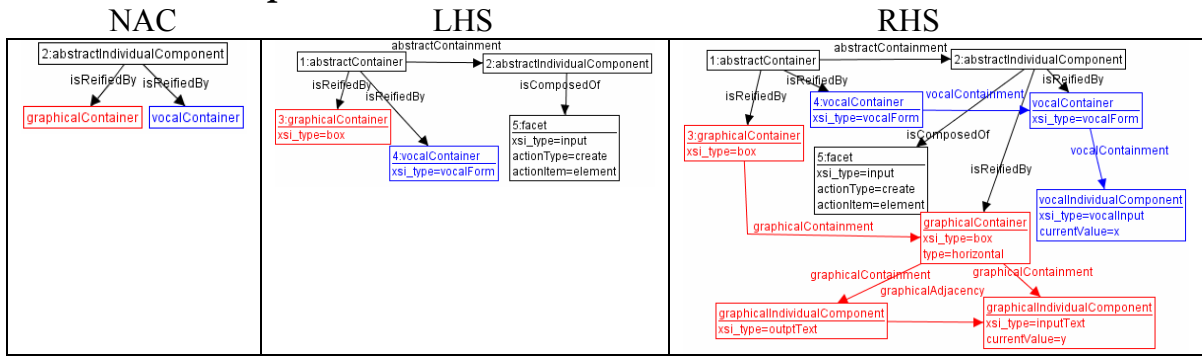
13. Transformation rules for simple output:

13.1. Vocal:



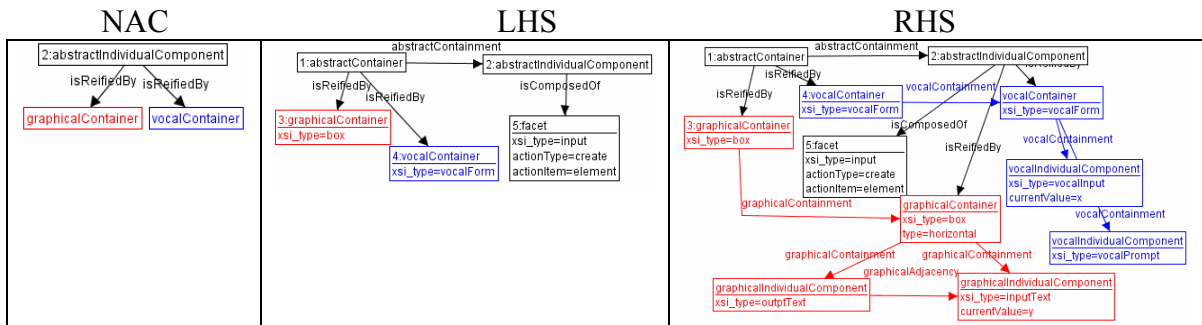
Rule 62. Generation of vocalPrompt components

13.2. Graphical:



Rule 63. Generation of graphical prompt (outputText) components

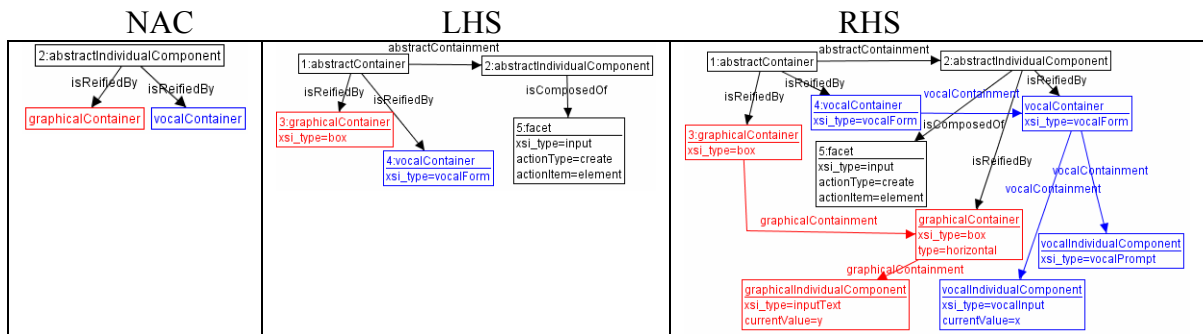
13.3. Multimodal



Rule 64. Generation of multimodal prompt (vocalPrompt and outputText) components

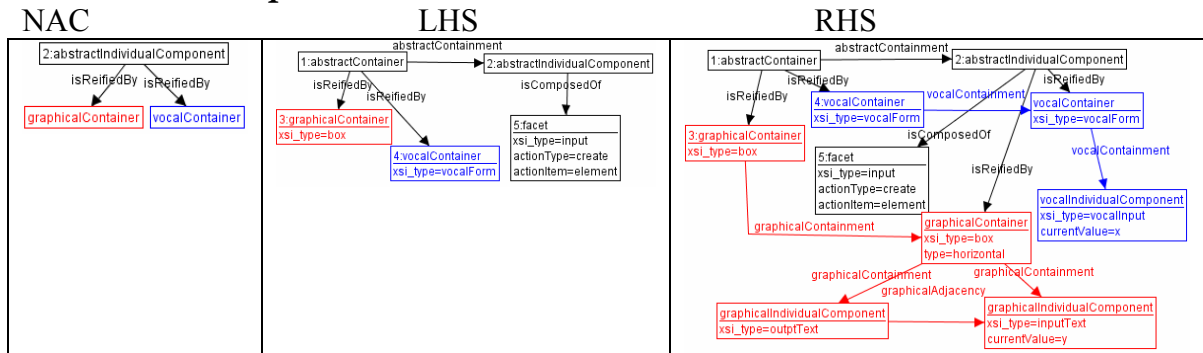
14. Transformation rules for prompting:

14.1. Vocal



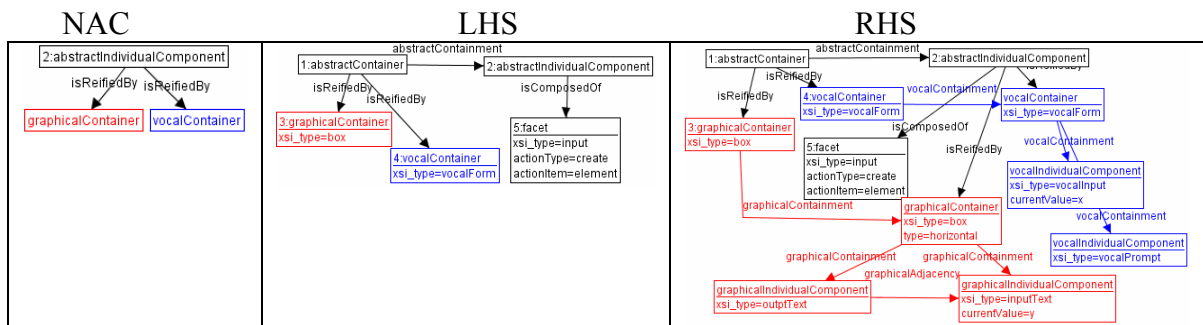
Rule 65. Generation of vocalPrompt components

14.2. Graphic



Rule 66. Generation of graphical prompt (outputText) components

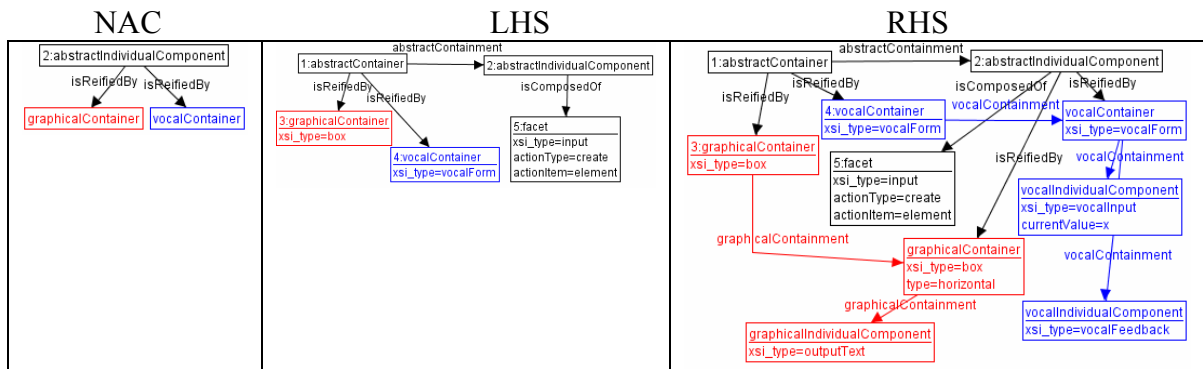
14.3. Multimodal



Rule 67. Generation of multimodal prompt (vocalPrompt and outputText) components

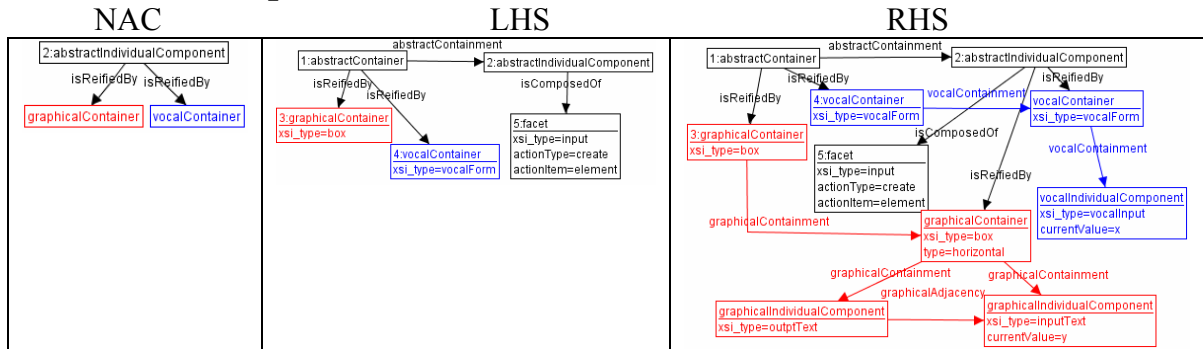
15. Transformation rules for immediate feedback

15.1. Vocal



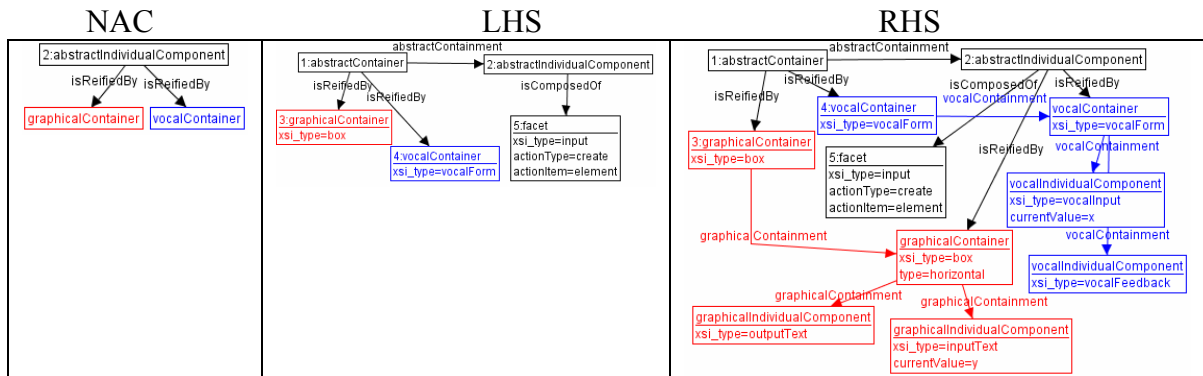
Rule 68. Generation of vocalFeedback components

15.2. Graphical



Rule 69. Generation of inputText components that will ensure the graphical feedback

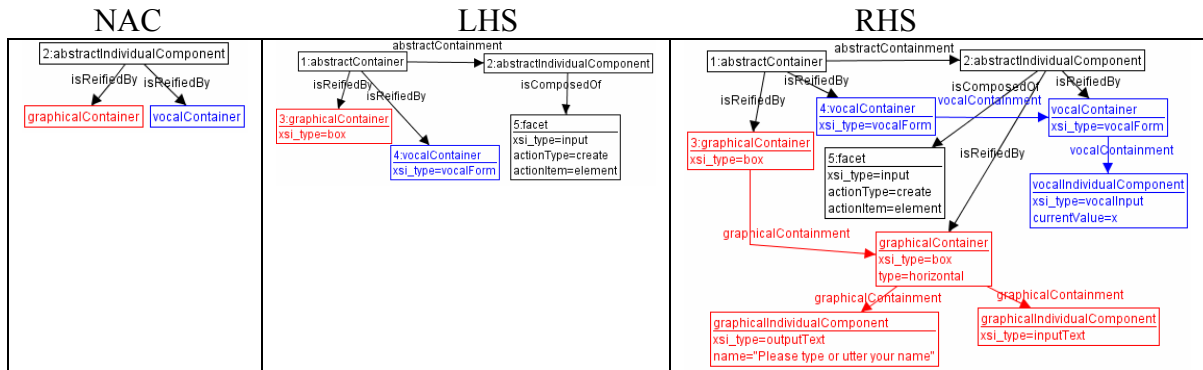
15.3. Multimodal



Rule 70. Generation of vocalFeedback and inputText components that will ensure the multimodal feedback

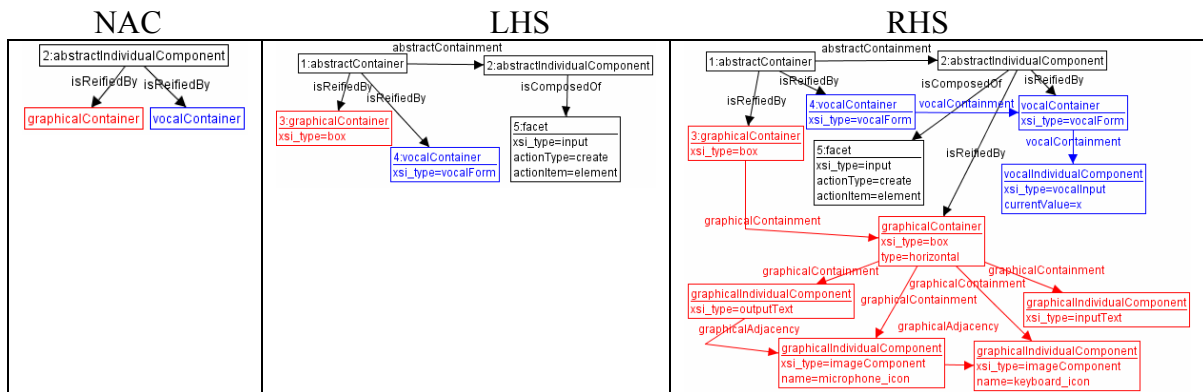
16. Transformation rules for guidance for input

16.1. Textual



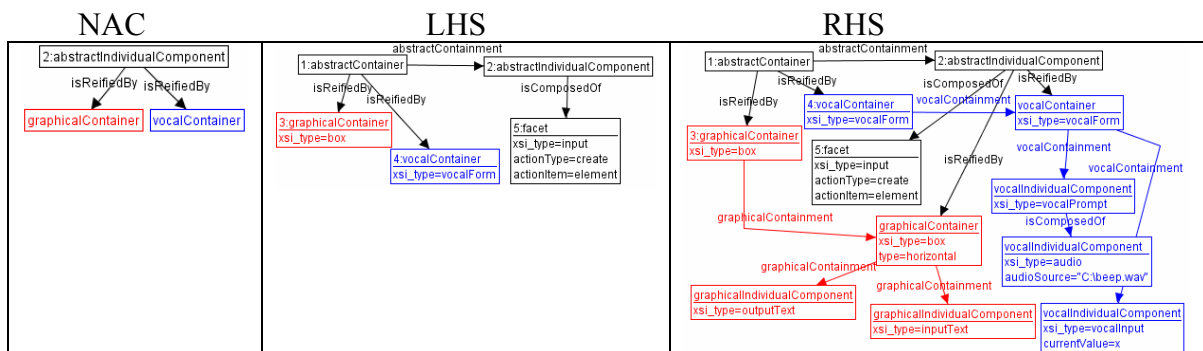
Rule 71. Generation of outputText components that ensure a textual guidance for input

16.2. Iconic



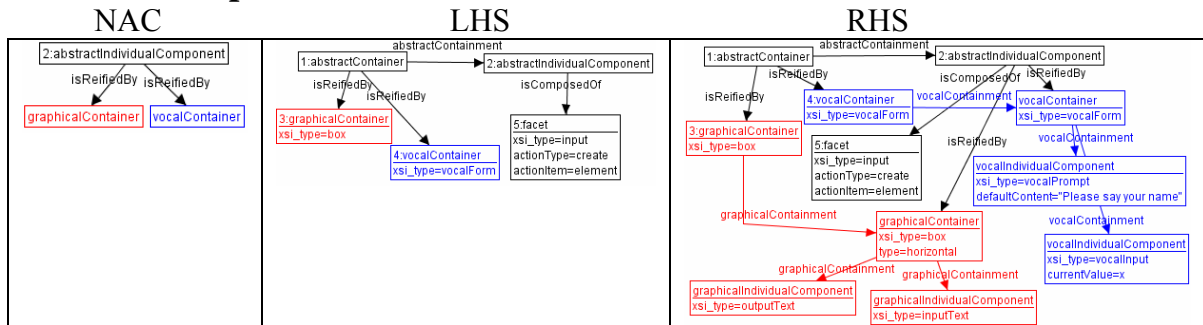
Rule 72. Generation of imageComponents that ensure an iconic guidance for input

16.3. Acoustic



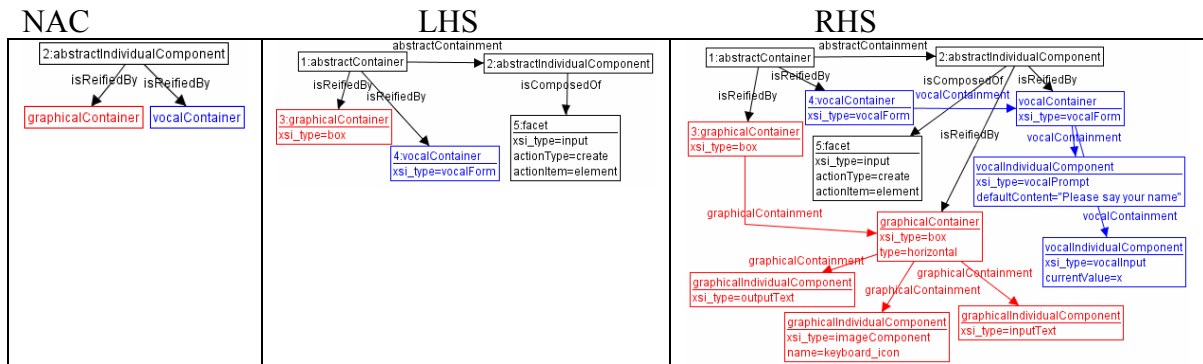
Rule 73. Generation of vocalPrompts that play an audio file in order to ensure an acoustic guidance for input

16.4. Speech



Rule 74. Generation of vocalPrompts synthesizing speech to ensure speech guidance for input

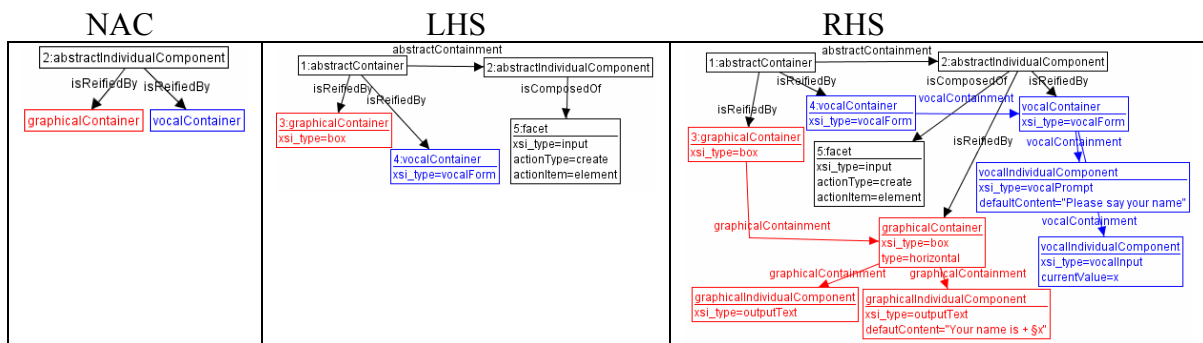
16.5. Multimodal



Rule 75. Generation of vocalPrompts synthesizing speech and of imageComponents to ensure multimodal (speech and iconic) guidance for input

17. Transformation rules for guidance for immediate feedback

17.1. Textual



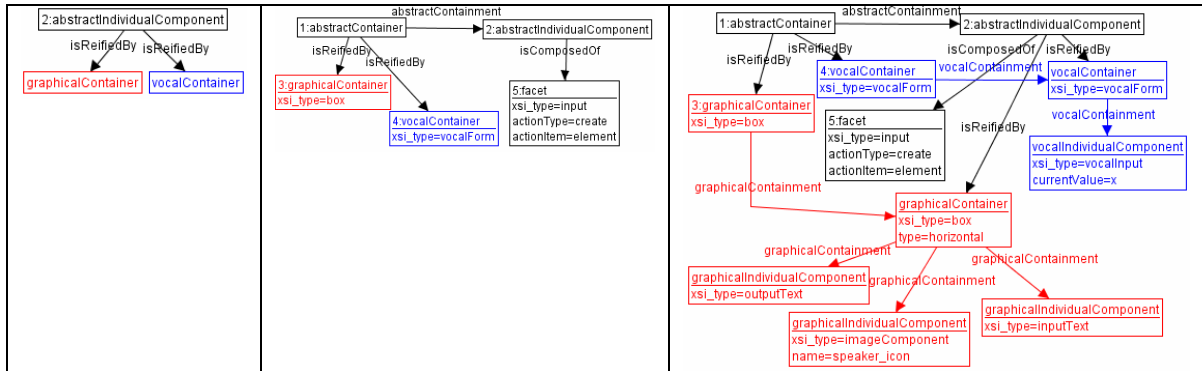
Rule 76. Generation of outputText components that ensure the textual guidance for feedback

17.2. Iconic

NAC

LHS

RHS



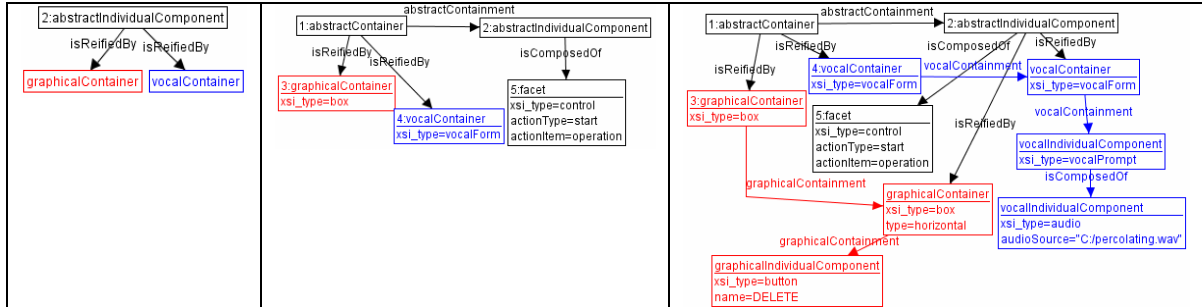
Rule 77. Generation of imageComponents that ensure the iconic guidance for feedback

17.3. Acoustic

NAC

LHS

RHS



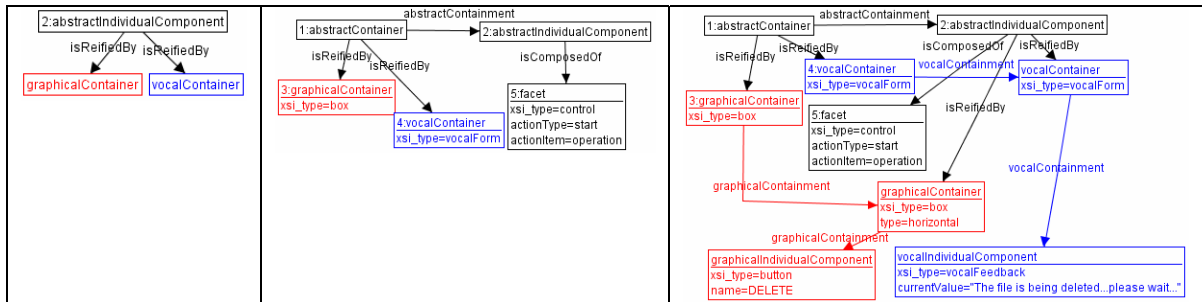
Rule 78. Generation of vocalPrompt components that play an audio file in order to ensure acoustic guidance for feedback

17.4. Speech

NAC

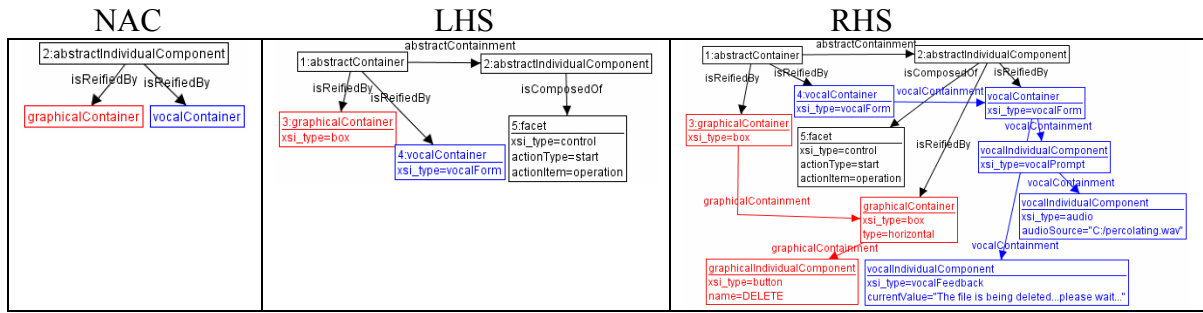
LHS

RHS



Rule 79. Generation of vocalFeedback components synthesizing speech that ensure speech guidance for feedback

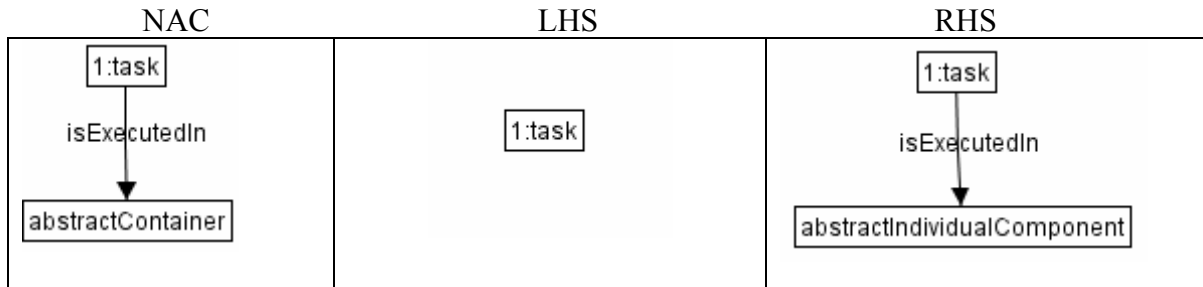
17.5. Multimodal



Rule 80. Generation of `vocalFeedback` and `vocalPrompt` that ensure multimodal (acoustic and speech) guidance for feedback

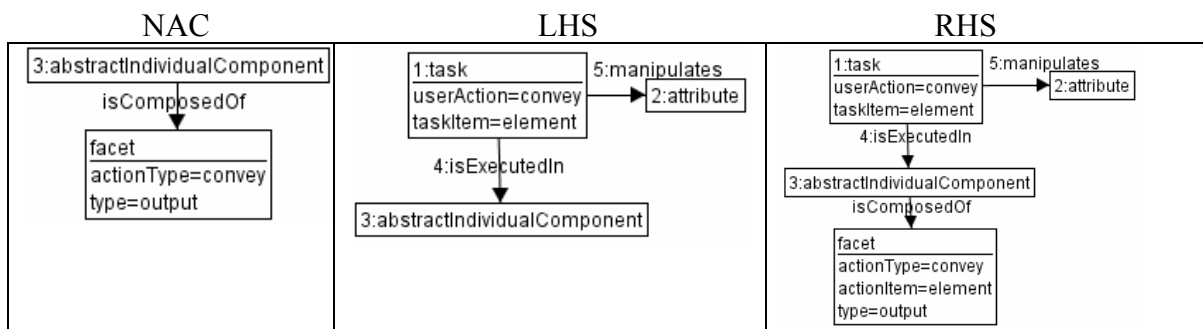
II. Additional transformation rules

1. Transformation rules for the identification of AUI structure



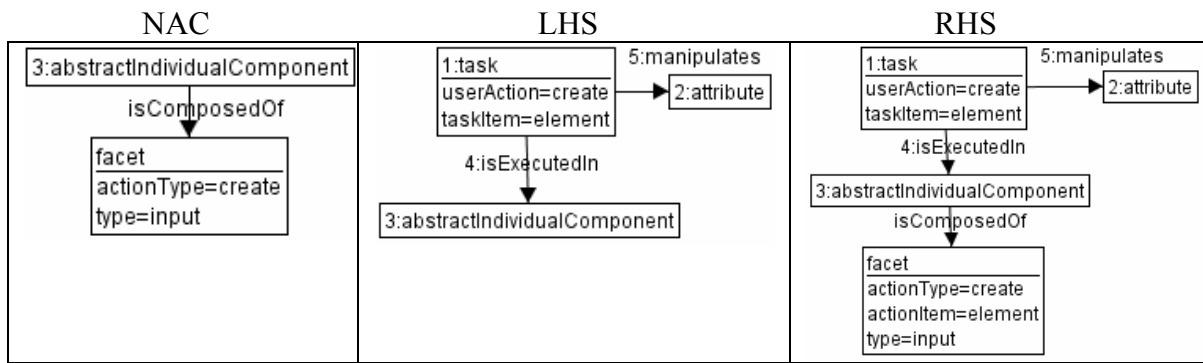
Rule 81. Create an AIC for leaf tasks

2. Transformation rules for selection of AICs

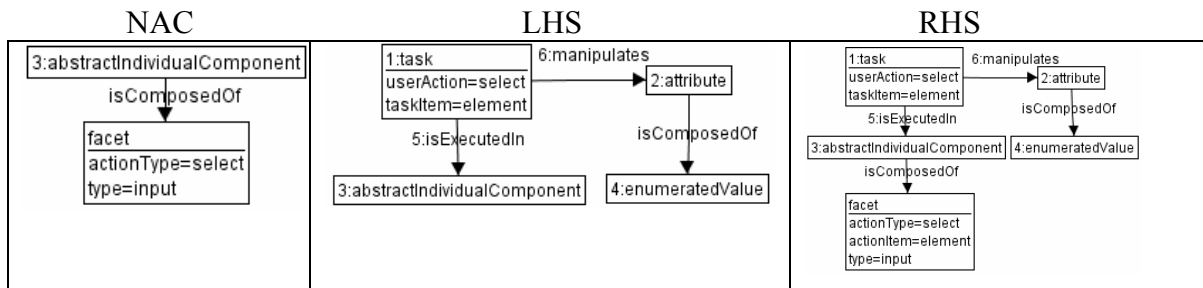


Rule 82. Create an output facet that conveys an element

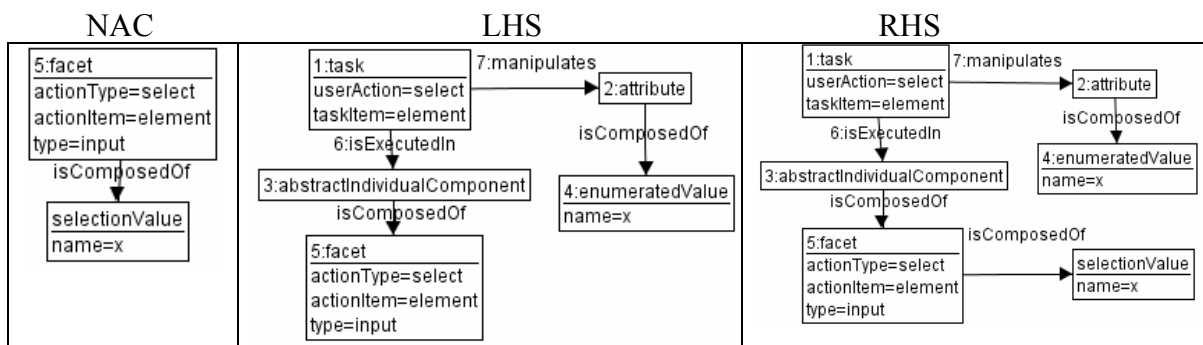
Appendix B Transformation rule catalog



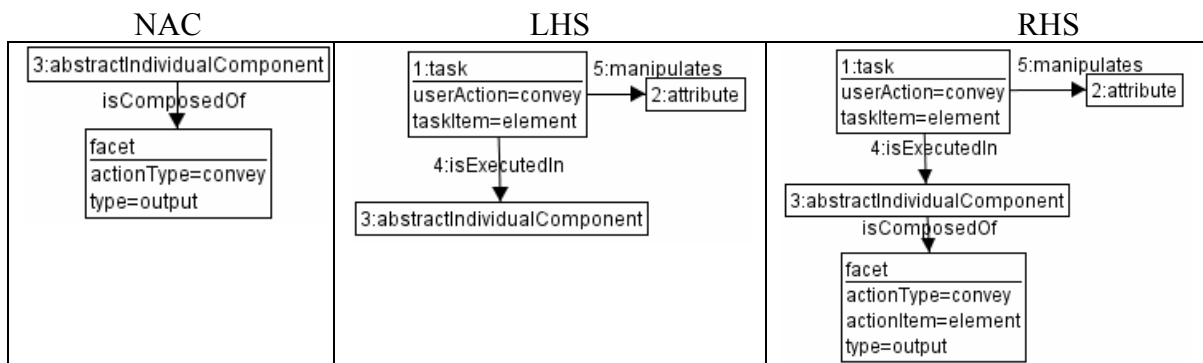
Rule 83. Create an input facet for AIC executed in tasks of type create



Rule 84. Create an input facet of type select element when an enumerated value attribute is encountered

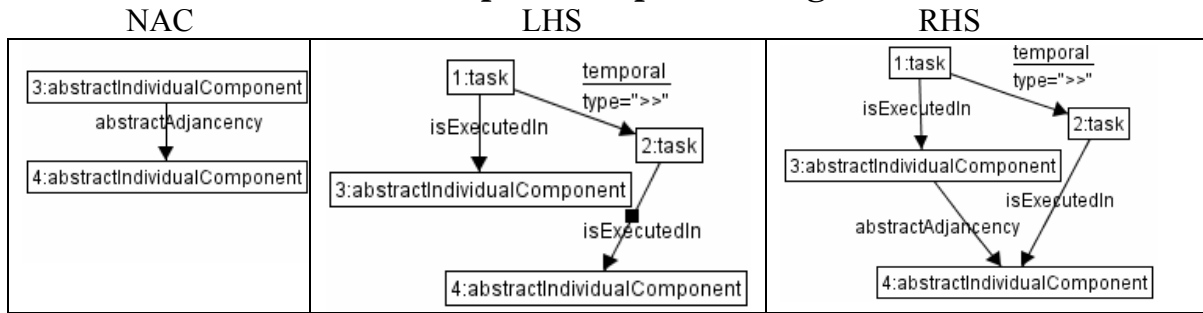


Rule 85. Create selection values for facets of type select for each enumerated value of an attribute

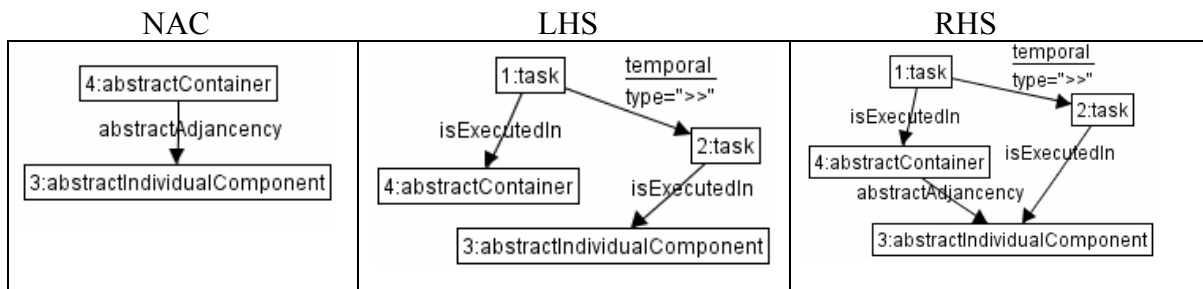


Rule 86. Create an output facet that conveys an element

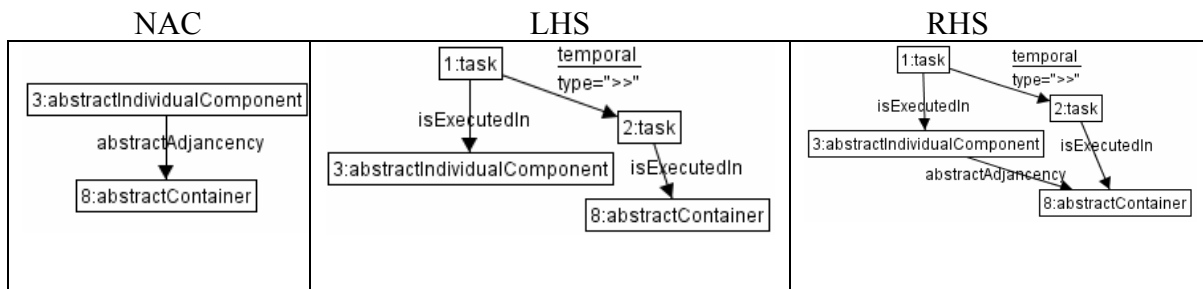
3. Transformation rules for spatio-temporal arrangement of AIOs



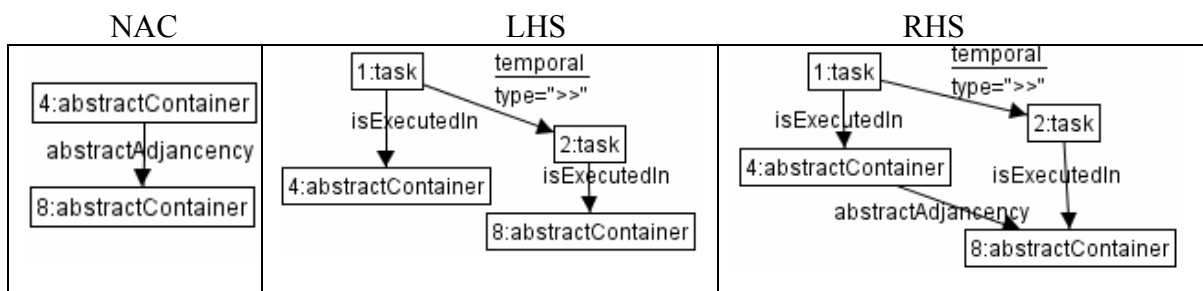
Rule 87. Generation of Abstract Adjacency relationship between <AIC, AIC> couples



Rule 88. Generation of Abstract Adjacency relationship between <AC, AIC> couples

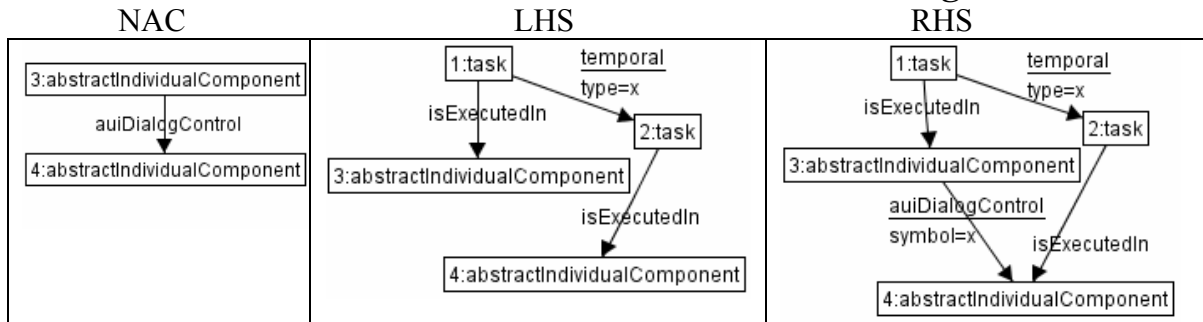


Rule 89. Generation of Abstract Adjacency relationship between <AIC, AC> couples

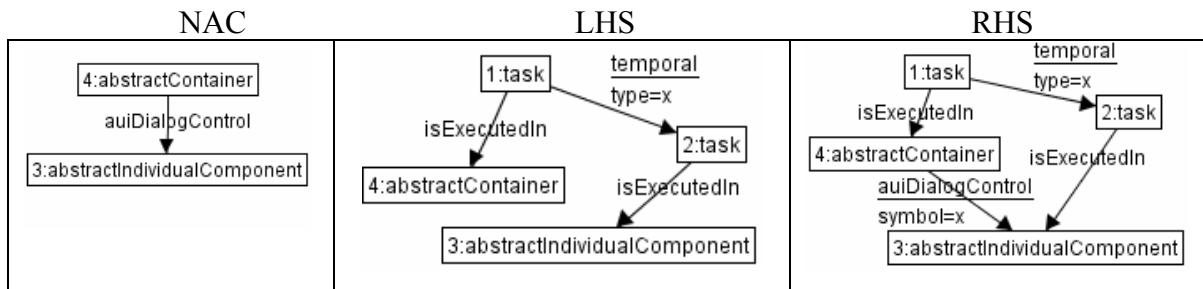


Rule 90. Generation of Abstract Adjacency relationship between <AC, AC> couples

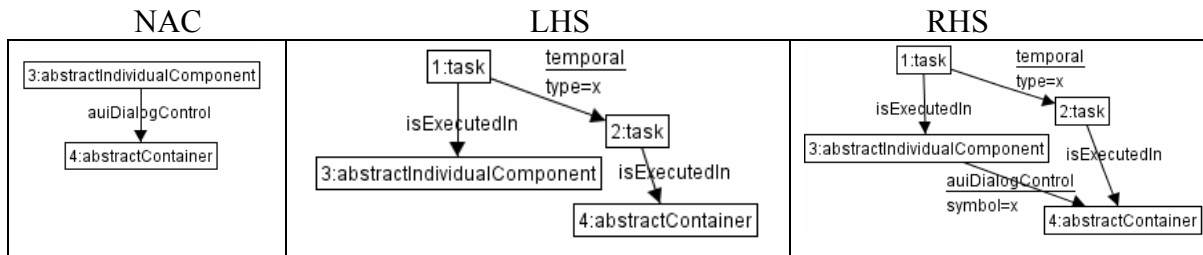
4. Transformation rules for the definition of abstract dialog control



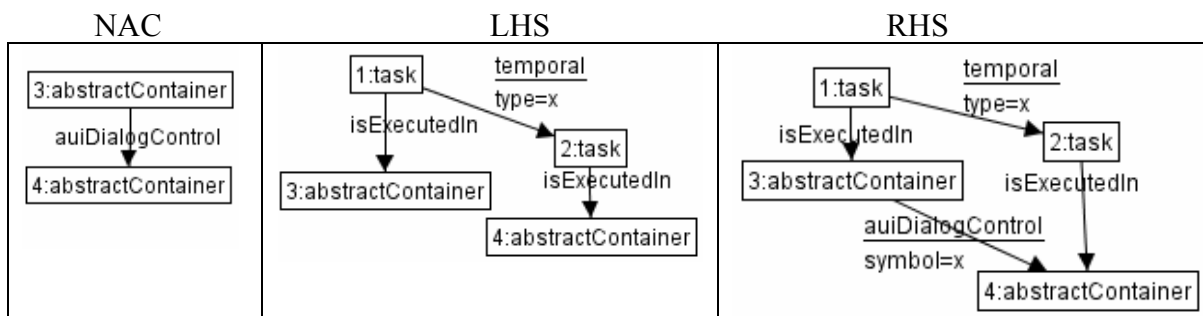
Rule 91. Generation of Abstract Dialog Control relationship between <AIC, AIC> couples



Rule 92. Generation of Abstract Dialog Control between <AC, AIC> couples

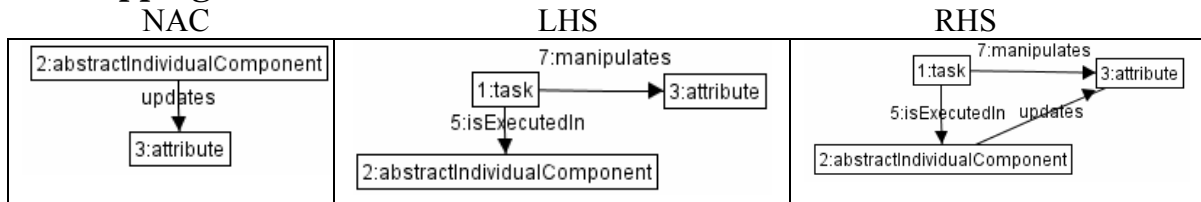


Rule 93. Generation of Abstract Dialog Control between <AIC, AC> couples

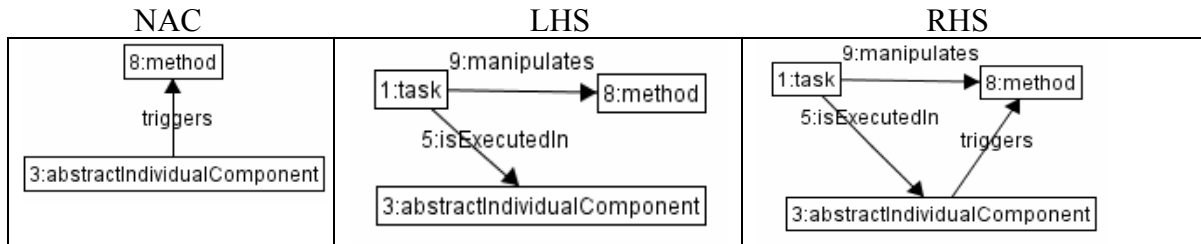


Rule 94. Generation of Abstract Dialog Control between <AC, AC> couples

5. Transformation rules for the derivation of the AUI to domain mappings

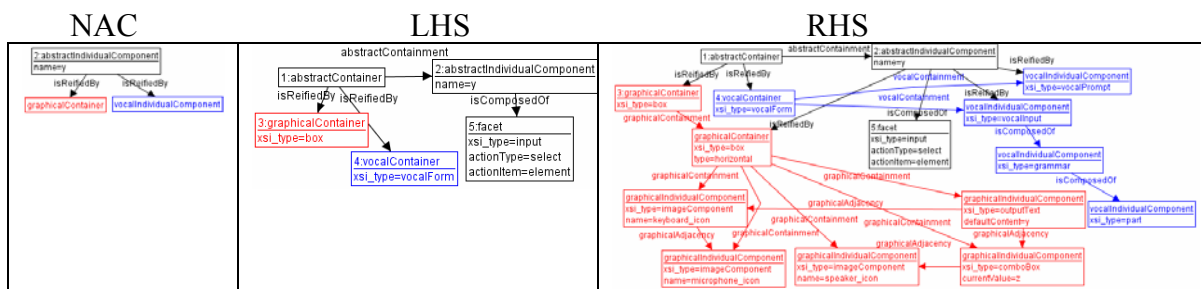


Rule 95. Generation of updates relationships for AICs

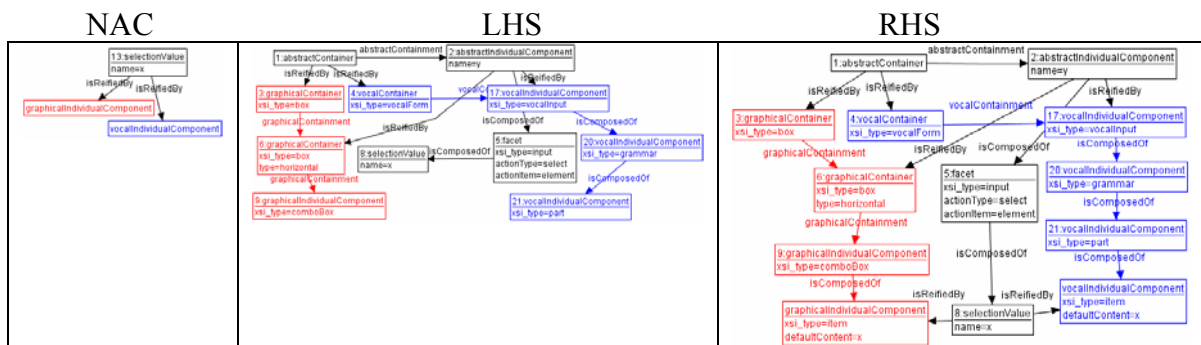


Rule 96. Generation of trigger relationships for AICs

6. Transformation rules for the selection of CICs

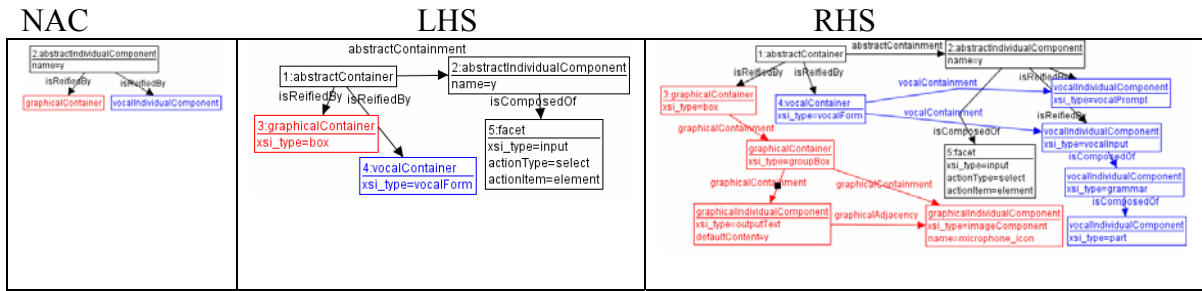


Rule 97. Generation of containers that will embed multimodal comboBox items

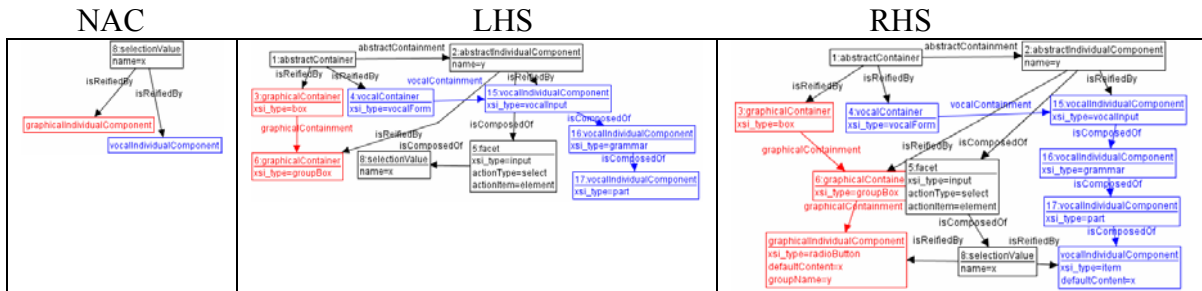


Rule 98. Generation of comboBox items and grammar items for each selection value of a facet of type select

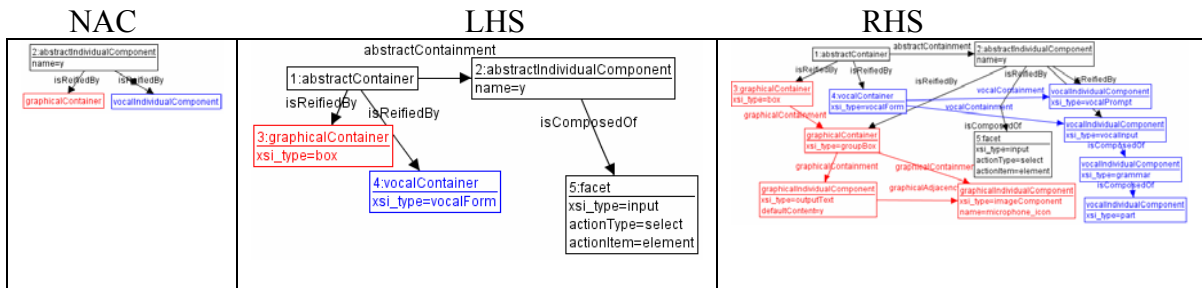
Appendix B Transformation rule catalog



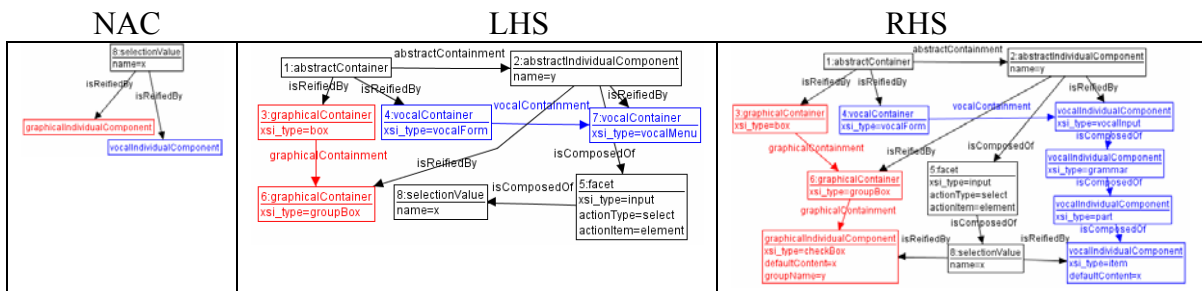
Rule 99. Generation of containers that will embed multimodal radioButtons



Rule 100. Generation of radioButtons and grammar items for each selection value of a facet of type select

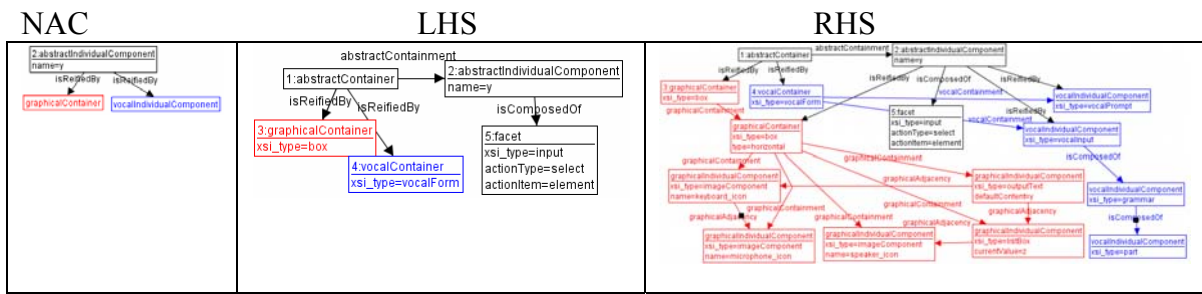


Rule 101. Generation of containers that will embed multimodal checkBoxes

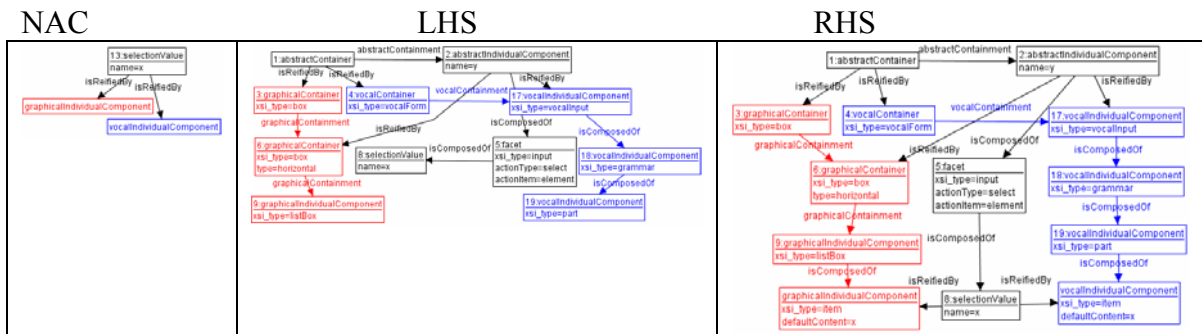


Rule 102. Generation of checkBoxes and grammar items for each selection value of a facet of type select

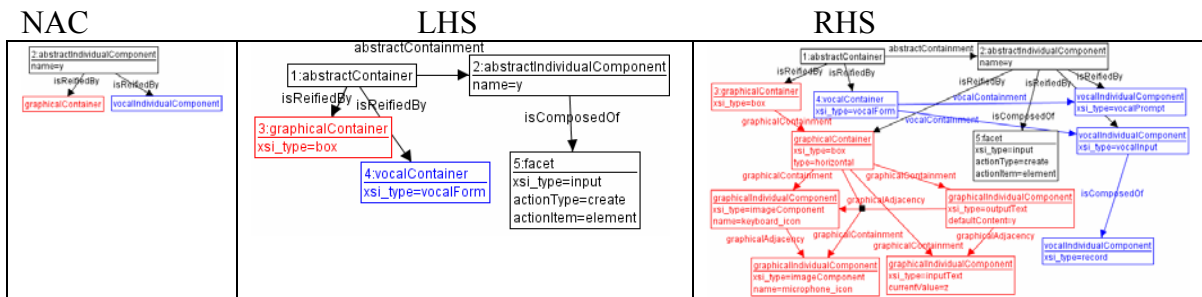
Appendix B Transformation rule catalog



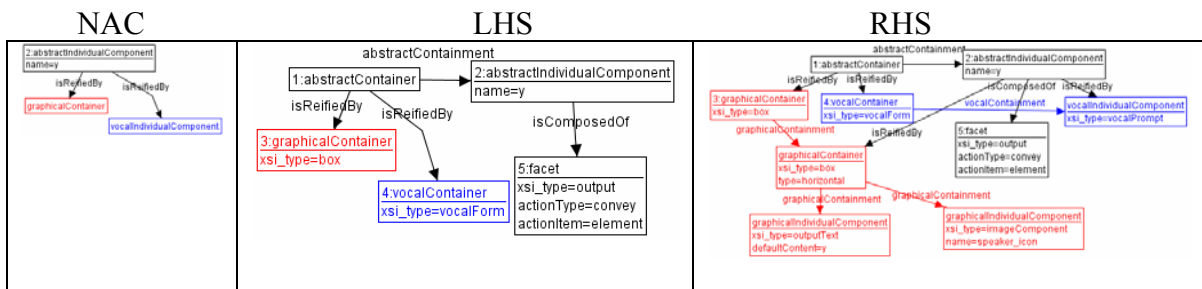
Rule 103. Generation of containers that will embed multimodal listBox items



Rule 104. Generation of listBox items and grammar items for each selection value of a facet of type select

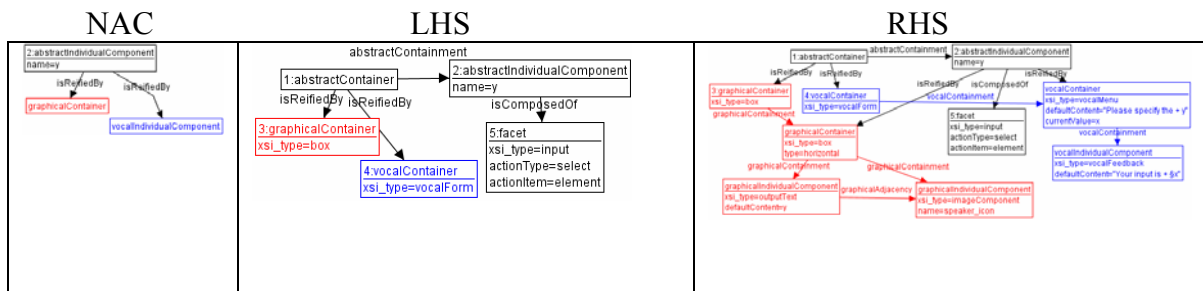


Rule 105. Generation of a multimodal inputText

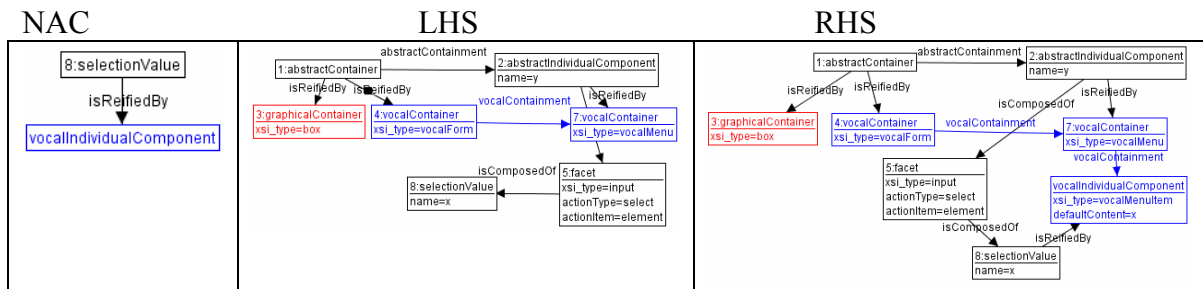


Rule 106. Generation of a multimodal outputText

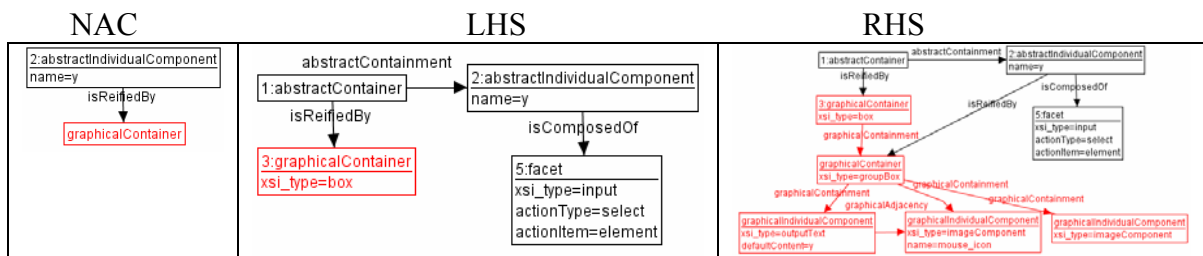
Appendix B Transformation rule catalog



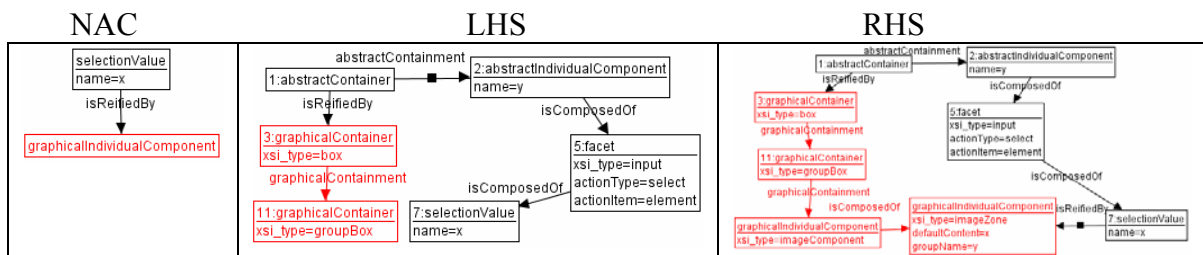
Rule 107. Generation of outputText and vocalMenu with feedback



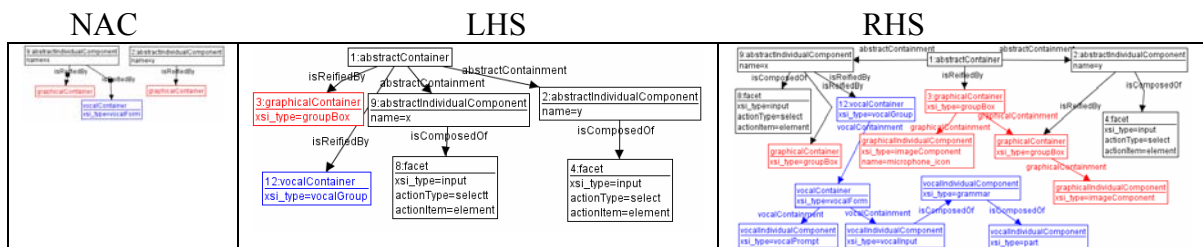
Rule 108. Generation of vocalMenuItems for each selection value of an input facet of type select



Rule 109. Generation of graphical containers embedding imageComponent elements

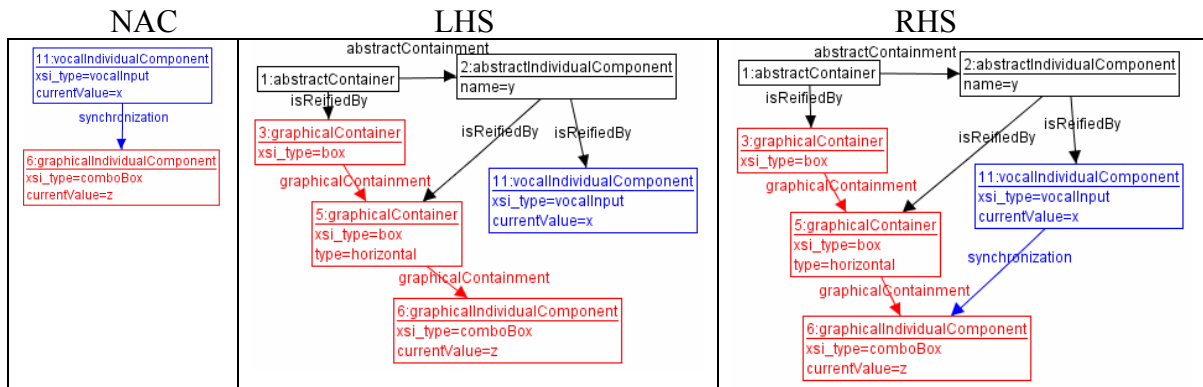


Rule 110. Generation of imageZone elements

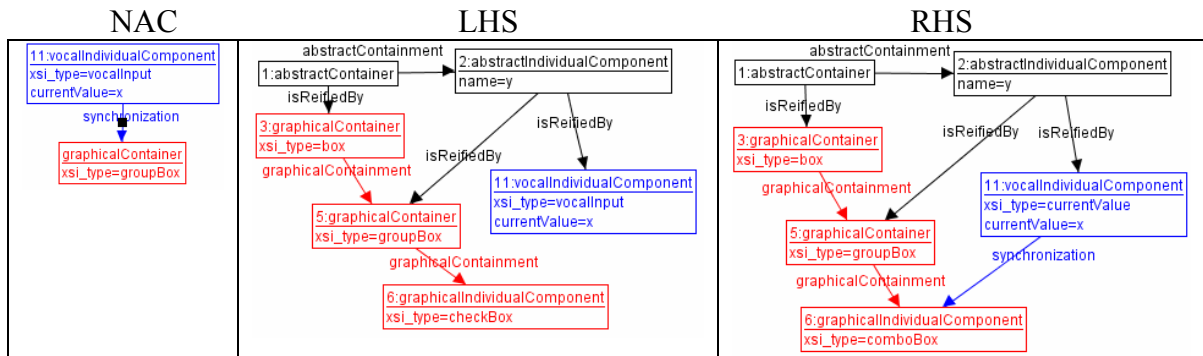


Rule 111. Generation of graphical and vocal containers to support the vocal instruction

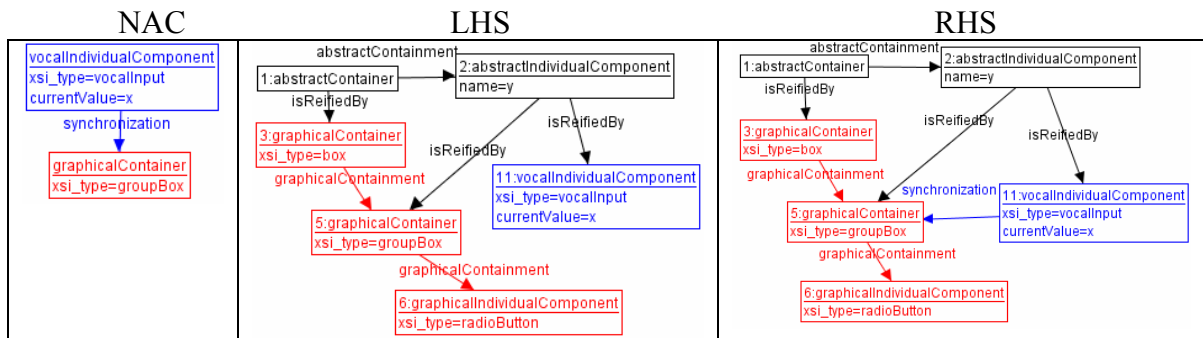
7. Transformation rules for the synchronization of CICs



Rule 116. Synchronization between a vocalInput and a comboBox

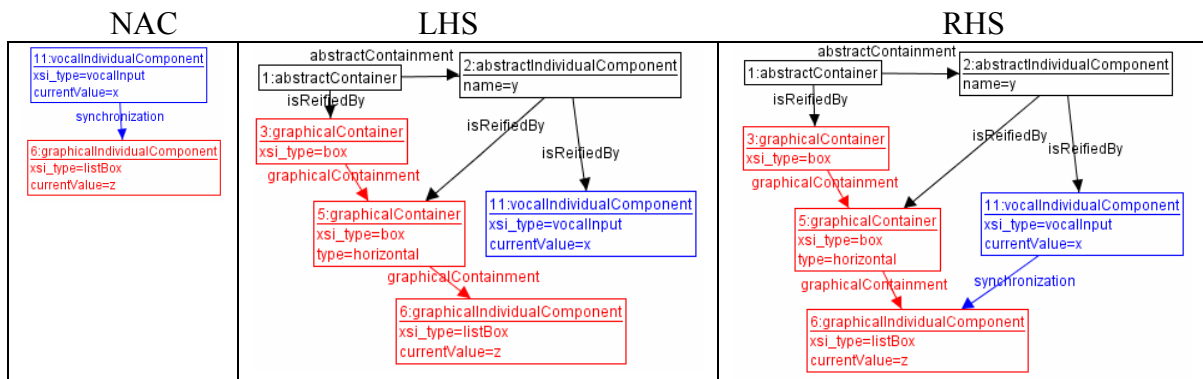


Rule 117. Synchronization between a vocalInput and a groupBox that embeds a set of check boxes

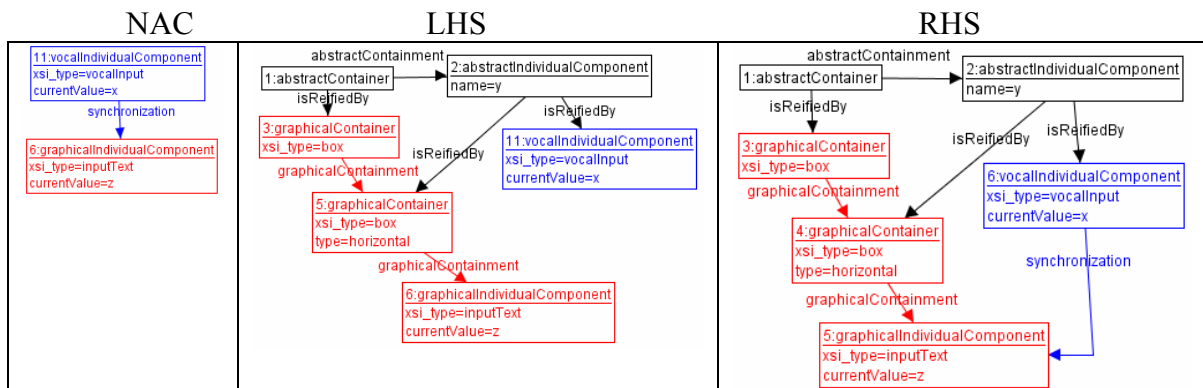


Rule 118. Synchronization between a vocalInput and a groupBox that embeds a set of radioButtons

Appendix B Transformation rule catalog

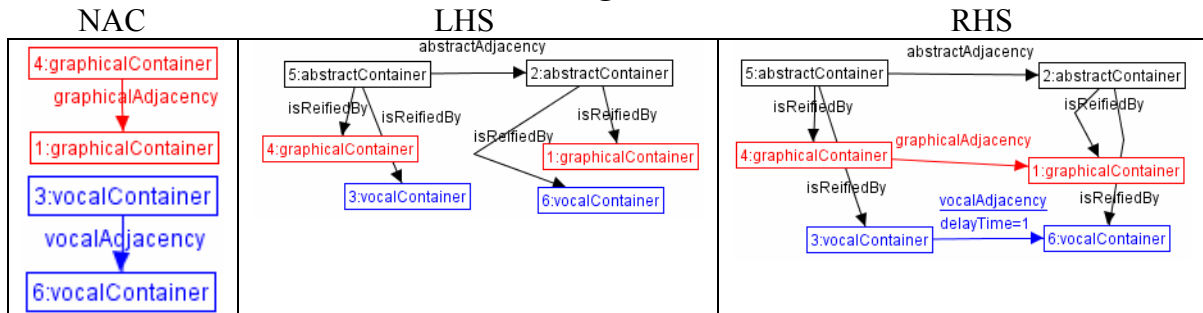


Rule 119. Synchronization between a vocalInput and a listBox



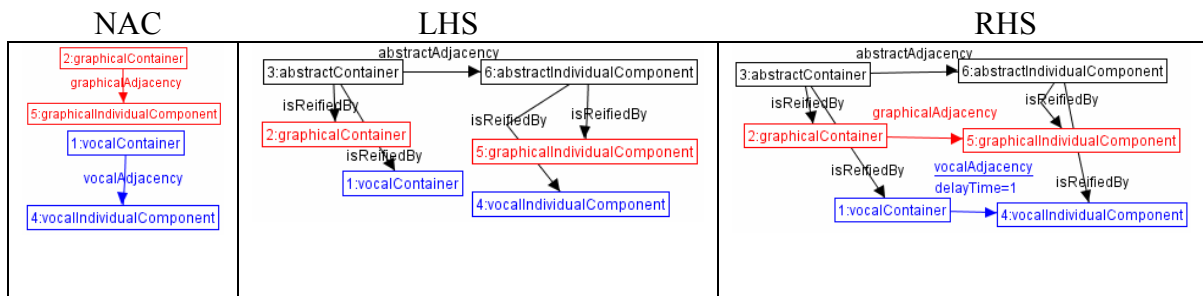
Rule 120. Synchronization between a vocalInput and an inputText

8. Transformation rules for the arrangement of CICs

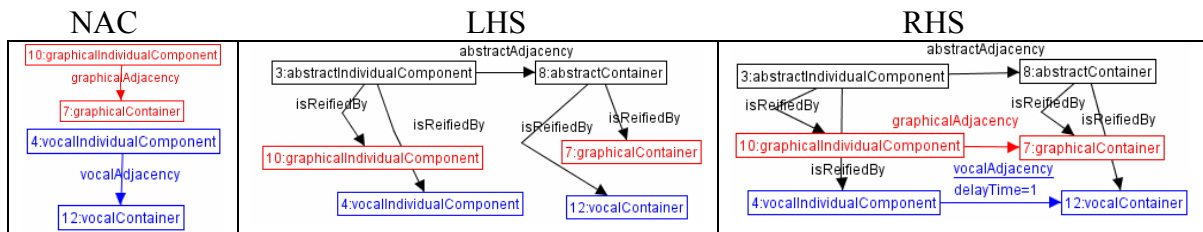


Rule 121. Generation of Concrete Adjacency relationship for <CC, CC> couples

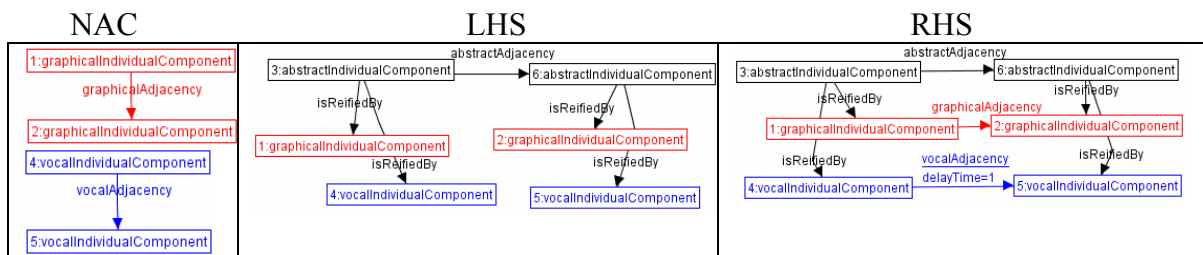
Appendix B Transformation rule catalog



Rule 122. Generation of Concrete Adjacency relationship for <CC, CIC>ouples

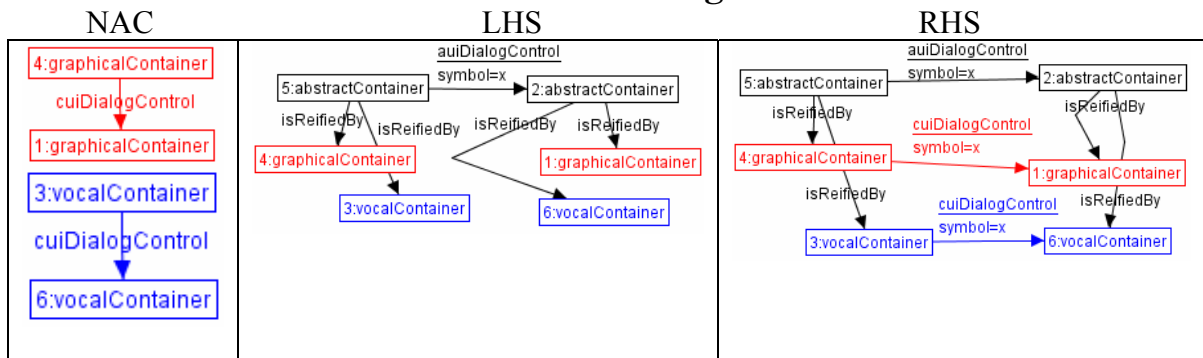


Rule 123. Generation of Concrete Adjacency relationship for <CIC, CC>ouples



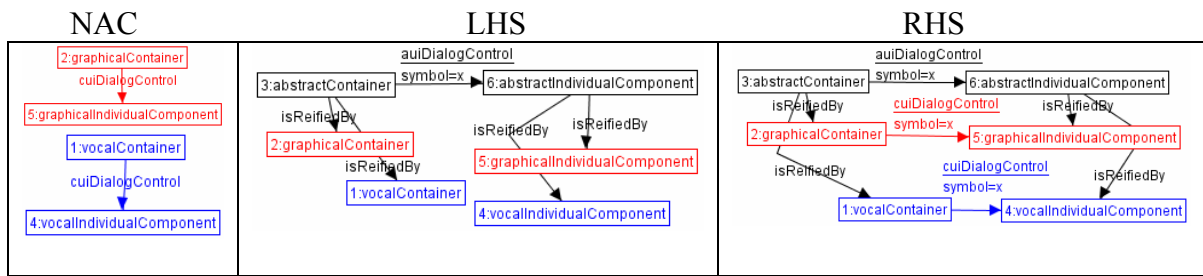
Rule 124. Generation of Concrete Adjacency relationship for <CIC, CIC>ouples

9. Transformation rules for Concrete Dialog Control definition

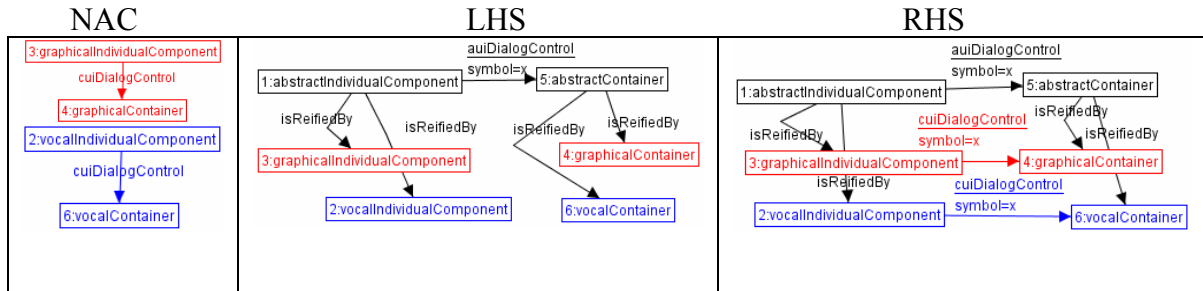


Rule 125. Rule Generation of Concrete Dialog Control Relationship for <CC, CC>ouples

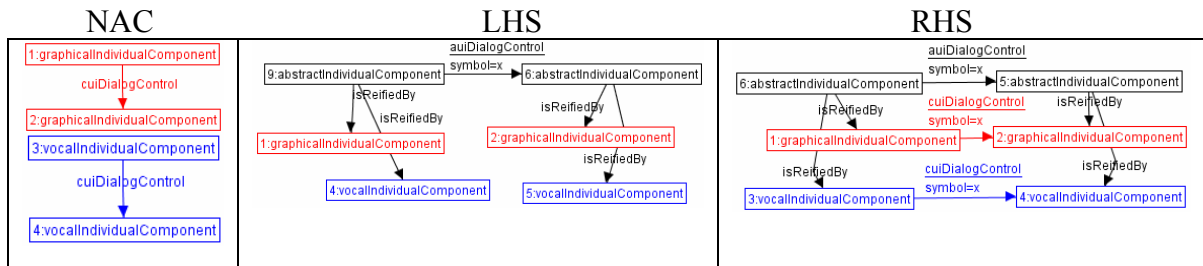
Appendix B Transformation rule catalog



Rule 126. Generation Concrete Dialog Control Relationship for <CC, CIC> couples

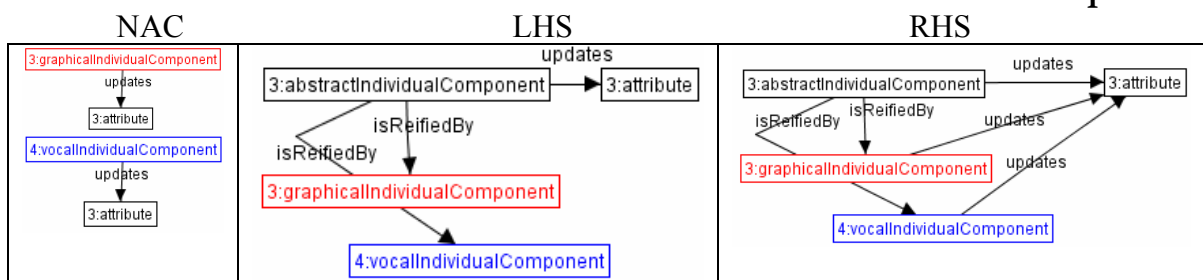


Rule 127. Generation Concrete Dialog Control Relationship for <CIC, CC> couples

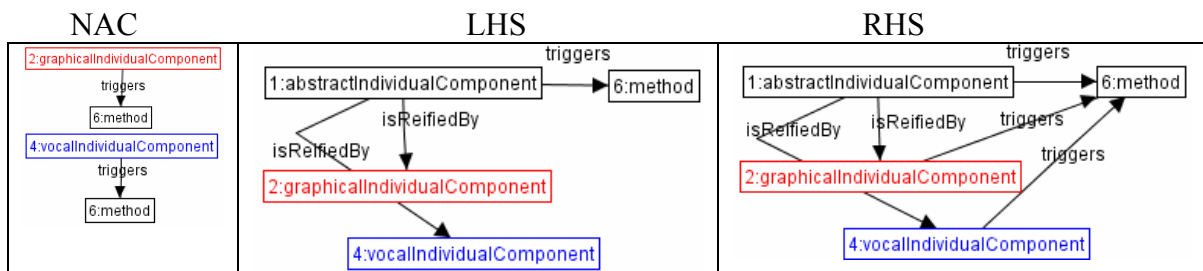


Rule 128. Generation of Concrete Dialog Control Relationship for <CIC, CIC> couples

10. Transformation rules for derivation of CUI to domain relationship



Rule 129. Transposition of *update* relationship



Rule 130. Transposition of *trigger* relationship

Appendix C. UsiXML concrete syntax for the specification of different combinations of input and output modalities

LABEL: due to the fact that a *label* widget does not suppose any input from the user, only the output interactions are considered (i.e., graphical, vocal and multimodal with redundancy in output):

- *Graphical interaction:*

```
<box id="b1" name="Box 1"...>
  <outputText id="OT1" name="Output 1" defaultContent="Welcome to the UCL site".../>
</box>
```

- *Vocal interaction:*

```
<vocalForm id="VF1" name="Form 1"...>
  <vocalPrompt id="VP1" name="Prompt 1" defaultContent="Welcome to the UCL site".../>
</vocalForm>
```

- *MM with redundancy in output:*

```
<vocalForm id="VF1" name="Form 1"...>
  <vocalPrompt id="VP1" name="Prompt 1" defaultContent="Welcome to the UCL site".../>
</vocalForm>
```

```
<box id="b1" name="Box 1"...>
  <outputText id="OT1" name="Output 1" defaultContent="Welcome to the UCL site".../>
  <imageComponent id="IC1" name="speaker_icon" defaultContent="speaker.jpg".../>
</box>
```

LABEL + COMBO BOX:

- *Graphical interaction:*

```
<box id="b1" name="Box 1"...>
  <outputText id="OT1" name="Output 1" defaultContent="Card type".../>
  <imageComponent id="IC3" name="mouse_icon" defaultContent="mouse.jpg".../>
  <comboBox id="CB1" name="Combo 1" isEnabled="true" currentValue="$x"...>
    <item id="IT1" name="Item 1" defaultContent="VISA".../>
    <item id="IT2" name="Item 2" defaultContent="MASTERCARD".../>
    <item id="IT3" name="Item 3" defaultContent="AMERICAN EXPRESS".../>
  </comboBox>
</box>
```

- *MM with G assignment in input and redundancy in output*

```
<box id="b1" name="Box 1"...>
  <outputText id="OT1" name="Output 1" defaultContent="Card type".../>
  <imageComponent id="IC3" name="mouse_icon" defaultContent="mouse.jpg".../>
  <comboBox id="CB1" name="Combo 1" isEnabled="true" currentValue="$x"...>
    <item id="IT1" name="Item 1" defaultContent="VISA".../>
    <item id="IT2" name="Item 2" defaultContent="MASTERCARD".../>
    <item id="IT3" name="Item 3" defaultContent="AMERICAN EXPRESS".../>
  </comboBox>
</box>
```

Appendix C: UsiXML concrete syntax for the specification of different combinations of input and output modalities

```
</comboBox>
<imageComponent id="IC4" name="speaker_icon" defaultContent="speaker.jpg" .../>
</box>

<vocalGroup id="VG1" name="Group 1" ...>
  <vocalFeedback id="F1" name="Feedback 1" defaultContent="Your choice is $x" .../>
</vocalGroup>

<synchronization>
  <source sourceId="F1"/>
  <target targetId="CB1"/>
</synchronization>
```

▪ *Vocal interaction:*

```
<vocalForm id="VF1" name="Form 1" ...>
  <vocalPrompt id="VP1" name="Prompt 1" defaultContent="Select the credit card type. Choose between VISA, MASTERCARD, AMERICAN EXPRESS" .../>
  <vocalInput id="VI1" name="Input 1" currentValue="$y">
    <grammar id="GR1" name="Grammar 1" ...>
      <part id="P1" name="Part 1" structure="choice" ...>
        <item id="IT1" name="Item 1" defaultContent="VISA" .../>
        <item id="IT2" name="Item 2" defaultContent="MASTERCARD" .../>
        <item id="IT3" name="Item 3" defaultContent="AMERICAN EXPRESS" .../>
      </part>
    </grammar>
  </vocalInput>
</vocalForm>
```

▪ *MM with V assignement in input and G assignement in output:*

```
<box id="b1" name="Box 1" ...>
  <outputText id="OT1" name="Output 1" defaultContent="Card type" .../>
  <imageComponent id="IC3" name="microphone_icon" defaultContent="microphone.jpg" .../>
  <comboBox id="CB1" name="Combo 1" isEnabled="false" currentValue="$x" ...>
    <item id="IT1" name="Item 1" defaultContent="VISA" .../>
    <item id="IT2" name="Item 2" defaultContent="MASTERCARD" .../>
    <item id="IT3" name="Item 3" defaultContent="AMERICAN EXPRESS" .../>
  </comboBox>
</box>
```

```
<vocalForm id="VF1" name="Form 1" ...>
  <vocalPrompt id="VP1" name="Prompt 1" defaultContent="Select the credit card type. Choose between VISA, MASTERCARD, AMERICAN EXPRESS" .../>
  <vocalInput id="VI1" name="Input 1" currentValue="$y">
    <grammar id="GR1" name="Grammar 1" ...>
      <part id="P1" name="Part 1" structure="choice" ...>
        <item id="IT1" name="Item 1" defaultContent="VISA" .../>
        <item id="IT2" name="Item 2" defaultContent="MASTERCARD" .../>
        <item id="IT3" name="Item 3" defaultContent="AMERICAN EXPRESS" .../>
      </part>
    </grammar>
  </vocalInput>
</vocalForm>
```

```
<synchronization>
  <source sourceId="VI1"/>
  <target targetId="CB1"/>
</synchronization>
```

▪ *MM with V assignement in input and redundancy in output*

```
<box id="b1" name="Box 1" ...>
  <outputText id="OT1" name="Output 1" defaultContent="Card type" .../>
  <imageComponent id="IC3" name="microphone_icon" defaultContent="microphone.jpg" .../>
  <comboBox id="CB1" name="Combo 1" isEnabled="false" currentValue="$x" ...>
```

Appendix C: UsiXML concrete syntax for the specification of different combinations of input and output modalities

```

    <item id="IT1" name="Item 1" defaultContent="VISA"../>
    <item id="IT2" name="Item 2" defaultContent="MASTERCARD"../>
    <item id="IT3" name="Item 3" defaultContent="AMERICAN EXPRESS"../>
  </comboBox>
  <imageComponent id="IC4" name="speaker_icon" defaultContent="speaker.jpg"../>
</box>

<vocalForm id="VF1" name="Form 1" ...>
  <vocalPrompt id="VP1" name="Prompt 1" defaultContent="Select the credit card type. Choose between VISA, MASTERCARD, AMERICAN EXPRESS"../>
  <vocalInput id="VI1" name="Input 1" currentValue="$y"../>
    <grammar id="GR1" name="Grammar 1" ...>
      <part id="P1" name="Part 1" structure="choice" ...>
        <item id="IT1" name="Item 1" defaultContent="VISA"../>
        <item id="IT2" name="Item 2" defaultContent="MASTERCARD"../>
        <item id="IT3" name="Item 3" defaultContent="AMERICAN EXPRESS"../>
      </part>
    </grammar>
  </vocalInput>
  <vocalFeedback id="F1" name="Feedback 1" defaultContent="Your choice is $y"../>
</vocalForm>

<synchronization>
  <source sourceId="VI1"../>
  <target targetId="CB1"../>
</synchronization>

▪ MM with equivalence in input and G assignement in output
<box id="b1" name="Box 1" ...>
  <outputText id="OT1" name="Output 1" defaultContent="Card type"../>
  <imageComponent id="IC3" name="microphone_icon" defaultContent="microphone.jpg"../>
  <imageComponent id="IC4" name="mouse_icon" defaultContent="mouse.jpg"../>
  <comboBox id="CB1" name="Combo 1" isEnabled="true" currentValue="$x" ...>
    <item id="IT1" name="Item 1" defaultContent="VISA"../>
    <item id="IT2" name="Item 2" defaultContent="MASTERCARD"../>
    <item id="IT3" name="Item 3" defaultContent="AMERICAN EXPRESS"../>
  </comboBox>
</box>

<vocalForm id="VF1" name="Form 1" ...>
  <vocalPrompt id="VP1" name="Prompt 1" defaultContent="Select the credit card type. Choose between VISA, MASTERCARD, AMERICAN EXPRESS"../>
  <vocalInput id="VI1" name="Input 1" currentValue="$y"../>
    <grammar id="GR1" name="Grammar 1" ...>
      <part id="P1" name="Part 1" structure="choice" ...>
        <item id="IT1" name="Item 1" defaultContent="VISA"../>
        <item id="IT2" name="Item 2" defaultContent="MASTERCARD"../>
        <item id="IT3" name="Item 3" defaultContent="AMERICAN EXPRESS"../>
      </part>
    </grammar>
  </vocalInput>
</vocalForm>

<synchronization>
  <source sourceId="VI1"../>
  <target targetId="CB1"../>
</synchronization>

▪ MM with equivalence in input and redundancy in output
<box id="b1" name="Box 1" ...>
  <outputText id="OT1" name="Output 1" defaultContent="Card type"../>
  <imageComponent id="IC3" name="microphone_icon" defaultContent="microphone.jpg"../>
  <imageComponent id="IC4" name="mouse_icon" defaultContent="mouse.jpg"../>
  <comboBox id="CB1" name="Combo 1" isEnabled="true" currentValue="$x" ...>

```

Appendix C: UsiXML concrete syntax for the specification of different combinations of input and output modalities

```
<item id="IT1" name="Item 1" defaultContent="VISA"../>
<item id="IT2" name="Item 2" defaultContent="MASTERCARD"../>
<item id="IT3" name="Item 3" defaultContent="AMERICAN EXPRESS"../>
</comboBox>
<imageComponent id="IC4" name="speaker_icon" defaultContent="speaker.jpg"../>
</box>

<vocalForm id="VF1" name="Form 1"...>
  <vocalPrompt id="VP1" name="Prompt 1" defaultContent="Select the credit card type. Choose between VISA, MASTERCARD, AMERICAN EXPRESS"../>
  <vocalInput id="VI1" name="Input 1" currentValue="$y"/>
    <grammar id="GR1" name="Grammar 1"...>
      <part id="P1" name="Part 1" structure="choice"...>
        <item id="IT1" name="Item 1" defaultContent="VISA"../>
        <item id="IT2" name="Item 2" defaultContent="MASTERCARD"../>
        <item id="IT3" name="Item 3" defaultContent="AMERICAN EXPRESS"../>
      </part>
    </grammar>
  </vocalInput>
  <vocalFeedback id="F1" name="Feedback 1" defaultContent="Your choice is $y"../>
</vocalForm>

<synchronization>
  <source sourceId="VI1"/>
  <target targetId="CB1"/>
</synchronization>
```

GROUP OF RADIO BUTTONS:

- *Graphical interaction:*

```
<box id="b1" name="Box 1"...>
  <groupBox id="GB1" name="Gender" currentValue="$x"...>
    <imageComponent id="IC3" name="mouse_icon" defaultContent="mouse.jpg"../>
    <radioButton id="RB1" name="Radio 1" groupName="Gender" defaultContent="Male" default-
    State="true" isEnabled="true"...>
    <radioButton id="RB2" name="Radio 2" groupName="Gender" defaultContent="Female" de-
    faultState="false" isEnabled="true"...>
  </groupBox>
</box>
```
- *MM with G assignment in input and redundancy in output:*

```
<box id="b1" name="Box 1"...>
  <groupBox id="GB1" name="Gender" currentValue="$x"...>
    <imageComponent id="IC3" name="mouse_icon" defaultContent="mouse.jpg"../>
    <radioButton id="RB1" name="Radio 1" groupName="Gender" defaultContent="Male" default-
    State="true" isEnabled="true"...>
    <radioButton id="RB2" name="Radio 2" groupName="Gender" defaultContent="Female" de-
    faultState="false" isEnabled="true"...>
  </groupBox>
  <imageComponent id="IC4" name="speaker_icon" defaultContent="speaker.jpg"../>
</box>

<vocalGroup id="VG1" name="Group 1"...>
  <vocalFeedback id="F1" name="Feedback 1" defaultContent="Your choice is $x"../>
</vocalGroup>

<synchronization>
  <source sourceId="F1"/>
  <target targetId="GB1"/>
</synchronization>
```
- *Vocal interaction:*

```
<vocalForm id="VF1" name="Form 1"...>
```

Appendix C: UsiXML concrete syntax for the specification of different combinations of input and output modalities

```

<vocalPrompt id="VP1" name="Prompt 1" defaultContent="Please say your gender. Choose between male and female" .../>
<vocalInput id="VI1" name="Input 1" currentValue="$y">
  <grammar id="GR1" name="Grammar 1" ...>
    <part id="P1" name="Part 1" structure="choice" ...>
      <item id="IT1" name="Item 1" defaultContent="male" .../>
      <item id="IT2" name="Item 2" defaultContent="female" .../>
    </part>
  </grammar>
</vocalInput>
</vocalForm>

```

- *MM with V assignement in input and G assignement in output:*

```

<box id="b1" name="Box 1" ...>
  <groupBox id="GB1" name="Gender" currentValue="$x" ...>
    <imageComponent id="IC3" name="microphone_icon" defaultContent="microphone.jpg" .../>
    <radioButton id="RB1" name="Radio 1" groupName="Gender" defaultContent="Male" defaultState="true" isEnabled="false" ...>
    <radioButton id="RB2" name="Radio 2" groupName="Gender" defaultContent="Female" defaultState="false" isEnabled="false" ...>
  </groupBox>
</box>

```

```

<vocalForm id="VF1" name="Form 1" ...>
  <vocalPrompt id="VP1" name="Prompt 1" defaultContent="Please say your gender. Choose between male and female" .../>
  <vocalInput id="VI1" name="Input 1" currentValue="$y">
    <grammar id="GR1" name="Grammar 1" ...>
      <part id="P1" name="Part 1" structure="choice" ...>
        <item id="IT1" name="Item 1" defaultContent="male" .../>
        <item id="IT2" name="Item 2" defaultContent="female" .../>
      </part>
    </grammar>
  </vocalInput>
</vocalForm>

```

```

<synchronization>
  <source sourceId="VI1"/>
  <target targetId="GB1"/>
</synchronization>

```

- *MM with V assignement in input and redundancy in output*

```

<box id="b1" name="Box 1" ...>
  <groupBox id="GB1" name="Gender" currentValue="$x" ...>
    <imageComponent id="IC3" name="microphone_icon" defaultContent="microphone.jpg" .../>
    <radioButton id="RB1" name="Radio 1" groupName="Gender" defaultContent="Male" defaultState="true" isEnabled="false" ...>
    <radioButton id="RB2" name="Radio 2" groupName="Gender" defaultContent="Female" defaultState="false" isEnabled="false" ...>
  </groupBox>
  <imageComponent id="IC4" name="speaker_icon" defaultContent="speaker.jpg" .../>
</box>

```

```

<vocalForm id="VF1" name="Form 1" ...>
  <vocalPrompt id="VP1" name="Prompt 1" defaultContent="Please say your gender. Choose between male and female" .../>
  <vocalInput id="VI1" name="Input 1" currentValue="$y">
    <grammar id="GR1" name="Grammar 1" ...>
      <part id="P1" name="Part 1" structure="choice" ...>
        <item id="IT1" name="Item 1" defaultContent="male" .../>
        <item id="IT2" name="Item 2" defaultContent="female" .../>
      </part>
    </grammar>
  </vocalInput>
</vocalForm>

```

Appendix C: UsiXML concrete syntax for the specification of different combinations of input and output modalities

```
</vocalInput>
<vocalFeedback id="F1" name="Feedback 1" defaultContent="Your choice is $y" .../>
</vocalForm>

<synchronization>
  <source sourceId="V11"/>
  <target targetId="GB1"/>
</synchronization>

▪ MM with equivalence in input and G assignement in output
<box id="b1" name="Box 1" ...>
  <outputText id="OT1" name="Output 1" defaultContent="Card type" .../>
  <groupBox id="GB1" name="Gender" currentValue="$x" ...>
    <imageComponent id="IC3" name="microphone_icon" defaultContent="microphone.jpg" .../>
    <imageComponent id="IC4" name="mouse_icon" defaultContent="mouse.jpg" .../>
    <radioButton id="RB1" name="Radio 1" groupName="Gender" defaultContent="Male" default-
State="true" isEnabled="true" ...>
    <radioButton id="RB2" name="Radio 2" groupName="Gender" defaultContent="Female" de-
faultState="false" isEnabled="true" ...>
  </groupBox>
</box>

<vocalForm id="VF1" name="Form 1" ...>
  <vocalPrompt id="VP1" name="Prompt 1" defaultContent="Please say your gender. Choose be-
tween male and female" .../>
  <vocalInput id="VI1" name="Input 1" currentValue="$y">
    <grammar id="GR1" name="Grammar 1" ...>
      <part id="P1" name="Part 1" structure="choice" ...>
        <item id="IT1" name="Item 1" defaultContent="male" .../>
        <item id="IT2" name="Item 2" defaultContent="female" .../>
      </part>
    </grammar>
  </vocalInput>
</vocalForm>

<synchronization>
  <source sourceId="V11"/>
  <target targetId="GB1"/>
</synchronization>

▪ MM with equivalence in input and redundancy in output
<box id="b1" name="Box 1" ...>
  <groupBox id="GB1" name="Gender" currentValue="$x" ...>
    <imageComponent id="IC3" name="microphone_icon" defaultContent="microphone.jpg" .../>
    <imageComponent id="IC4" name="mouse_icon" defaultContent="mouse.jpg" .../>
    <radioButton id="RB1" name="Radio 1" groupName="Gender" defaultContent="Male" default-
State="true" isEnabled="true" ...>
    <radioButton id="RB2" name="Radio 2" groupName="Gender" defaultContent="Female" de-
faultState="false" isEnabled="true" ...>
  </groupBox>
  <imageComponent id="IC4" name="speaker_icon" defaultContent="speaker.jpg" .../>
</box>

<vocalForm id="VF1" name="Form 1" ...>
  <vocalPrompt id="VP1" name="Prompt 1" defaultContent="Please say your gender. Choose be-
tween male and female" .../>
  <vocalInput id="VI1" name="Input 1" currentValue="$y">
    <grammar id="GR1" name="Grammar 1" ...>
      <part id="P1" name="Part 1" structure="choice" ...>
        <item id="IT1" name="Item 1" defaultContent="male" .../>
        <item id="IT2" name="Item 2" defaultContent="female" .../>
      </part>
    </grammar>
  </vocalInput>
</vocalForm>
```

Appendix C: UsiXML concrete syntax for the specification of different combinations of input and output modalities

```
</vocalInput>
<vocalFeedback id="F1" name="Feedback 1" defaultContent="Your choice is $y" .../>
</vocalForm>
```

```
<synchronization>
  <source sourceId="V11"/>
  <target targetId="GB1"/>
</synchronization>
```

GROUP OF CHECK BOXES:

▪ *Graphical interaction:*

```
<box id="b1" name="Box 1" ...>
  <groupBox id="GB1" name="Hobbies" currentValue="$x" ...>
    <imageComponent id="IC3" name="mouse_icon" defaultContent="mouse.jpg" .../>
    <checkBox id="CB1" name="Check 1" groupName="Hobbies" defaultContent="Sports" default-
State="true" isEnabled="true" ...>
    <checkBox id="CB2" name="Check 2" groupName="Hobbies" defaultContent="Travel" default-
State="false" isEnabled="true" ...>
    <checkBox id="CB3" name="Check 3" groupName="Hobbies" defaultContent="Music" default-
State="true" isEnabled="true" ...>
    <checkBox id="CB4" name="Check 4" groupName="Hobbies" defaultContent="Movies" default-
State="true" isEnabled="true" ...>
  </groupBox>
</box>
```

▪ *MM with G assignment in input and redundancy in output*

```
<box id="b1" name="Box 1" ...>
  <groupBox id="GB1" name="Hobbies" currentValue="$x" ...>
    <imageComponent id="IC3" name="mouse_icon" defaultContent="mouse.jpg" .../>
    <checkBox id="CB1" name="Check 1" groupName="Hobbies" defaultContent="Sports" default-
State="true" isEnabled="true" ...>
    <checkBox id="CB2" name="Check 2" groupName="Hobbies" defaultContent="Travel" default-
State="false" isEnabled="true" ...>
    <checkBox id="CB3" name="Check 3" groupName="Hobbies" defaultContent="Music" default-
State="true" isEnabled="true" ...>
    <checkBox id="CB4" name="Check 4" groupName="Hobbies" defaultContent="Movies" default-
State="true" isEnabled="true" ...>
  </groupBox>
  <imageComponent id="IC4" name="speaker_icon" defaultContent="speaker.jpg" .../>
</box>
```

```
<vocalGroup id="VG1" name="Group 1" ...>
  <vocalFeedback id="F1" name="Feedback 1" defaultContent="Your choice is $x" .../>
</vocalGroup>
```

```
<synchronization>
  <source sourceId="F1"/>
  <target targetId="GB1"/>
</synchronization>
```

▪ *Vocal interaction:*

```
<vocalForm id="VF1" name="Form 1" ...>
  <vocalPrompt id="VP1" name="Prompt 1" defaultContent="Please select your hobbies. Choose
among the following options: sports, travels, music, movies" .../>
  <vocalInput id="VI1" name="Input 1" currentValue="$y">
    <grammar id="GR1" name="Grammar 1" ...>
      <part id="P1" name="Part 1" structure="asynchronous" ...>
        <item id="IT1" name="Item 1" defaultContent="sports" .../>
        <item id="IT2" name="Item 2" defaultContent="travels" .../>
        <item id="IT1" name="Item 3" defaultContent="music" .../>
        <item id="IT2" name="Item 4" defaultContent="movies" .../>
      </part>
    </grammar>
  </vocalInput>
</vocalForm>
```


Appendix C: UsiXML concrete syntax for the specification of different combinations of input and output modalities

```

    </grammar>
  </vocalInput>
</vocalForm>

```

- *MM with V assignement in input and G assignement in output:*

```

<box id="b1" name="Box 1"...>
  <groupBox id="GB1" name="Hobbies" currentValue="$x"...>
    <imageComponent id="IC3" name="microphone_icon" defaultContent="microphone.jpg".../>
    <checkBox id="CB1" name="Check 1" groupName="Hobbies" defaultContent="Sports" default-
State="true" isEnabled="false" ...>
    <checkBox id="CB2" name="Check 2" groupName="Hobbies" defaultContent="Travel" default-
State="false" isEnabled="false" ...>
    <checkBox id="CB3" name="Check 3" groupName="Hobbies" defaultContent="Music" default-
State="true" isEnabled="false" ...>
    <checkBox id="CB4" name="Check 4" groupName="Hobbies" defaultContent="Movies" default-
State="true" isEnabled="false" ...>
  </groupBox>
</box>

```

```

<vocalForm id="VF1" name="Form 1"...>
  <vocalPrompt id="VP1" name="Prompt 1" defaultContent="Please select your hobbies. Choose
among the following options: sports, travels, music, movies".../>
  <vocalInput id="VI1" name="Input 1" currentValue="$y">
    <grammar id="GR1" name="Grammar 1"...>
      <part id="P1" name="Part 1" structure="asynchronous"...>
        <item id="IT1" name="Item 1" defaultContent="sports".../>
        <item id="IT2" name="Item 2" defaultContent="travels".../>
        <item id="IT1" name="Item 3" defaultContent="music".../>
        <item id="IT2" name="Item 4" defaultContent="movies".../>
      </part>
    </grammar>
  </vocalInput>
</vocalForm>

```

```

<synchronization>
  <source sourceId="VI1"/>
  <target targetId="GB1"/>
</synchronization>

```

- *MM with V assignement in input and redundancy in output:*

```

<box id="b1" name="Box 1"...>
  <groupBox id="GB1" name="Hobbies" currentValue="$x"...>
    <imageComponent id="IC3" name="microphone_icon" defaultContent="microphone.jpg".../>
    <checkBox id="CB1" name="Check 1" groupName="Hobbies" defaultContent="Sports" default-
State="true" isEnabled="false" ...>
    <checkBox id="CB2" name="Check 2" groupName="Hobbies" defaultContent="Travel" default-
State="false" isEnabled="false" ...>
    <checkBox id="CB3" name="Check 3" groupName="Hobbies" defaultContent="Music" default-
State="true" isEnabled="false" ...>
    <checkBox id="CB4" name="Check 4" groupName="Hobbies" defaultContent="Movies" default-
State="true" isEnabled="false" ...>
  </groupBox>
  <imageComponent id="IC4" name="speaker_icon" defaultContent="speaker.jpg".../>
</box>

```

```

<vocalForm id="VF1" name="Form 1"...>
  <vocalPrompt id="VP1" name="Prompt 1" defaultContent="Please select your hobbies. Choose
among the following options: sports, travels, music, movies".../>
  <vocalInput id="VI1" name="Input 1" currentValue="$y">
    <grammar id="GR1" name="Grammar 1"...>
      <part id="P1" name="Part 1" structure="asynchronous"...>
        <item id="IT1" name="Item 1" defaultContent="sports".../>
        <item id="IT2" name="Item 2" defaultContent="travels".../>
      </part>
    </grammar>
  </vocalInput>
</vocalForm>

```


Appendix C: UsiXML concrete syntax for the specification of different combinations of input and output modalities

```

        <item id="IT1" name="Item 3" defaultContent="music" .../>
        <item id="IT2" name="Item 4" defaultContent="movies" .../>
    </part>
</grammar>
</vocalInput>
<vocalFeedback id="F1" name="Feedback 1" defaultContent="Your choice is $y" .../>
</vocalForm>

<synchronization>
    <source sourceId="V11"/>
    <target targetId="GB1"/>
</synchronization>

```

- *MM with equivalence in input and G assignment in output*

```

<box id="b1" name="Box 1" ...>
    <outputText id="OT1" name="Output 1" defaultContent="Card type" .../>
    <imageComponent id="IC3" name="microphone_icon" defaultContent="microphone.jpg" .../>
    <imageComponent id="IC4" name="mouse_icon" defaultContent="mouse.jpg" .../>
    <groupBox id="GB1" name="Hobbies" currentValue="$x" ...>
        <imageComponent id="IC3" name="microphone_icon" defaultContent="microphone.jpg" .../>
        <imageComponent id="IC4" name="mouse_icon" defaultContent="mouse.jpg" .../>
        <checkBox id="CB1" name="Check 1" groupName="Hobbies" defaultContent="Sports" default-
State="true" isEnabled="true" ...>
        <checkBox id="CB2" name="Check 2" groupName="Hobbies" defaultContent="Travel" default-
State="false" isEnabled="true" ...>
        <checkBox id="CB3" name="Check 3" groupName="Hobbies" defaultContent="Music" default-
State="true" isEnabled="true" ...>
        <checkBox id="CB4" name="Check 4" groupName="Hobbies" defaultContent="Movies" default-
State="true" isEnabled="true" ...>
    </groupBox>
</box>

<vocalForm id="VF1" name="Form 1" ...>
    <vocalPrompt id="VP1" name="Prompt 1" defaultContent="Please select your hobbies. Choose
among the following options: sports, travels, music, movies" .../>
    <vocalInput id="VI1" name="Input 1" currentValue="$y">
        <grammar id="GR1" name="Grammar 1" ...>
            <part id="P1" name="Part 1" structure="asynchronous" ...>
                <item id="IT1" name="Item 1" defaultContent="sports" .../>
                <item id="IT2" name="Item 2" defaultContent="travels" .../>
                <item id="IT1" name="Item 3" defaultContent="music" .../>
                <item id="IT2" name="Item 4" defaultContent="movies" .../>
            </part>
        </grammar>
    </vocalInput>
</vocalForm>

<synchronization>
    <source sourceId="V11"/>
    <target targetId="GB1"/>
</synchronization>

```

- *MM with equivalence in input and redundancy in output/*

```

<box id="b1" name="Box 1" ...>
    <groupBox id="GB1" name="Hobbies" currentValue="$x" ...>
        <imageComponent id="IC3" name="microphone_icon" defaultContent="microphone.jpg" .../>
        <imageComponent id="IC4" name="mouse_icon" defaultContent="mouse.jpg" .../>
        <checkBox id="CB1" name="Check 1" groupName="Hobbies" defaultContent="Sports" default-
State="true" isEnabled="true" ...>
        <checkBox id="CB2" name="Check 2" groupName="Hobbies" defaultContent="Travel" default-
State="false" isEnabled="true" ...>
        <checkBox id="CB3" name="Check 3" groupName="Hobbies" defaultContent="Music" default-
State="true" isEnabled="true" ...>
    </groupBox>
</box>

```

Appendix C: UsiXML concrete syntax for the specification of different combinations of input and output modalities

```
<checkBox id="CB4" name="Check 4" groupName="Hobbies" defaultContent="Movies" default-
State="true" isEnabled="true" ...>
</groupBox>
<imageComponent id="IC4" name="speaker_icon" defaultContent="speaker.jpg" .../>
</box>

<vocalPrompt id="VP1" name="Prompt 1" defaultContent="Please select your hobbies. Choose among
the following options: sports, travels, music, movies" .../>
<vocalInput id="VI1" name="Input 1" currentValue="$y">
<grammar id="GR1" name="Grammar 1" ...>
<part id="P1" name="Part 1" structure="asynchronous" ...>
<item id="IT1" name="Item 1" defaultContent="sports" .../>
<item id="IT2" name="Item 2" defaultContent="travels" .../>
<item id="IT1" name="Item 3" defaultContent="music" .../>
<item id="IT2" name="Item 4" defaultContent="movies" .../>
</part>
</grammar>
</vocalInput>
<vocalFeedback id="F1" name="Feedback 1" defaultContent="Your choice is $y" .../>
</vocalForm>

<synchronization>
<source sourceId="VI1"/>
<target targetId="GB1"/>
</synchronization>
```

LABEL + LIST BOX:

- *Graphical interaction:*

```
<box id="b1" name="Box 1" ...>
<outputText id="OT1" name="Output 1" defaultContent="Singers" .../>
<imageComponent id="IC3" name="mouse_icon" defaultContent="mouse.jpg" .../>
<listBox id="LB1" name="List 1" isEnabled="true" currentValue="$x" ...>
<item id="IT1" name="Item 1" defaultContent="Chris Hay" .../>
<item id="IT2" name="Item 2" defaultContent="Lee Hardy" .../>
</listBox>
</box>
```

- *MM with G assignement in input and redundancy in output*

```
<box id="b1" name="Box 1" ...>
<outputText id="OT1" name="Output 1" defaultContent="Singers" .../>
<imageComponent id="IC3" name="mouse_icon" defaultContent="mouse.jpg" .../>
<listBox id="LB1" name="List 1" isEnabled="true" currentValue="$x" ...>
<item id="IT1" name="Item 1" defaultContent="Chris Hay" .../>
<item id="IT2" name="Item 2" defaultContent="Lee Hardy" .../>
</listBox>
<imageComponent id="IC4" name="speaker_icon" defaultContent="speaker.jpg" .../>
</box>
```

```
<vocalGroup id="VG1" name="Group 1" ...>
<vocalFeedback id="F1" name="Feedback 1" defaultContent="Your choice is $x" .../>
</vocalGroup>
```

```
<synchronization>
<source sourceId="F1"/>
<target targetId="LB1"/>
</synchronization>
```

- *Vocal interaction:*

```
<vocalForm id="VF1" name="Form 1" ...>
<vocalPrompt id="VP1" name="Prompt 1" defaultContent="Please choose your favourite singer:
Chris Hay, Lee Hardy, ..." .../>
<vocalInput id="VI1" name="Input 1" currentValue="$y"/>
```

Appendix C: UsiXML concrete syntax for the specification of different combinations of input and output modalities

```

    <grammar id="GR1" name="Grammar 1"...>
      <part id="P1" name="Part 1" structure="choice"...>
        <item id="IT1" name="Item 1" defaultContent="Chris Hay".../>
        <item id="IT2" name="Item 2" defaultContent="Lee Hardy".../>
      </part>
    </grammar>
  </vocalInput>
</vocalForm>

```

- *MM with V assignement in input and G assignement in output:*

```

<box id="b1" name="Box 1"...>
  <outputText id="OT1" name="Output 1" defaultContent="Singers".../>
  <imageComponent id="IC3" name="microphone_icon" defaultContent="microphone.jpg".../>
  <listBox id="LB1" name="List 1" isEnabled="true" currentValue="$x" ...>
    <item id="IT1" name="Item 1" defaultContent="Chris Hay".../>
    <item id="IT2" name="Item 2" defaultContent="Lee Hardy".../>
  </listBox>
</box>

<vocalForm id="VF1" name="Form 1"...>
  <vocalPrompt id="VP1" name="Prompt 1" defaultContent="Please choose your favourite singer:
Chris Hay, Lee Hardy, ...".../>
  <vocalInput id="VI1" name="Input 1" currentValue="$y"/>
    <grammar id="GR1" name="Grammar 1"...>
      <part id="P1" name="Part 1" structure="choice"...>
        <item id="IT1" name="Item 1" defaultContent="Chris Hay".../>
        <item id="IT2" name="Item 2" defaultContent="Lee Hardy".../>
      </part>
    </grammar>
  </vocalInput>
</vocalForm>

<synchronization>
  <source sourceId="VI1"/>
  <target targetId="LB1"/>
</synchronization>

```

- *MM with V assignement in input and redundancy in output:*

```

<box id="b1" name="Box 1"...>
  <outputText id="OT1" name="Output 1" defaultContent="Singers".../>
  <imageComponent id="IC3" name="microphone_icon" defaultContent="microphone.jpg".../>
  <listBox id="LB1" name="List 1" isEnabled="true" currentValue="$x" ...>
    <item id="IT1" name="Item 1" defaultContent="Chris Hay".../>
    <item id="IT2" name="Item 2" defaultContent="Lee Hardy".../>
  </listBox>
  <imageComponent id="IC4" name="speaker_icon" defaultContent="speaker.jpg".../>
</box>

<vocalForm id="VF1" name="Form 1"...>
  <vocalPrompt id="VP1" name="Prompt 1" defaultContent="Please choose your favourite singer:
Chris Hay, Lee Hardy, ...".../>
  <vocalInput id="VI1" name="Input 1" currentValue="$y"/>
    <grammar id="GR1" name="Grammar 1"...>
      <part id="P1" name="Part 1" structure="choice"...>
        <item id="IT1" name="Item 1" defaultContent="Chris Hay".../>
        <item id="IT2" name="Item 2" defaultContent="Lee Hardy".../>
      </part>
    </grammar>
  </vocalInput>
  <vocalFeedback id="F1" name="Feedback 1" defaultContent="Your choice is $y".../>
</vocalForm>

<synchronization>

```

Appendix C: UsiXML concrete syntax for the specification of different combinations of input and output modalities

```
<source sourceId="V11"/>
<target targetId="LB1"/>
</synchronization>
```

- *MM with equivalence in input and G assignement in output:*

```
<box id="b1" name="Box 1"...>
  <outputText id="OT1" name="Output 1" defaultContent="Singers" .../>
  <imageComponent id="IC3" name="microphone_icon" defaultContent="microphone.jpg" .../>
  <imageComponent id="IC4" name="mouse_icon" defaultContent="mouse.jpg" .../>
  <listBox id="LB1" name="List 1" isEnabled="true" currentValue="$x" ...>
    <item id="IT1" name="Item 1" defaultContent="Chris Hay" .../>
    <item id="IT2" name="Item 2" defaultContent="Lee Hardy" .../>
  </listBox>
</box>

<vocalForm id="VF1" name="Form 1" ...>
  <vocalPrompt id="VP1" name="Prompt 1" defaultContent="Please choose your favourite singer:
Chris Hay, Lee Hardy, ..." .../>
  <vocalInput id="V11" name="Input 1" currentValue="$y"/>
    <grammar id="GR1" name="Grammar 1" ...>
      <part id="P1" name="Part 1" structure="choice" ...>
        <item id="IT1" name="Item 1" defaultContent="Chris Hay" .../>
        <item id="IT2" name="Item 2" defaultContent="Lee Hardy" .../>
      </part>
    </grammar>
  </vocalInput>
</vocalForm>
```

```
<synchronization>
  <source sourceId="V11"/>
  <target targetId="LB1"/>
</synchronization>
```

- *MM with equivalence in input and redundancy in output*

```
<box id="b1" name="Box 1"...>
  <outputText id="OT1" name="Output 1" defaultContent="Singers" .../>
  <imageComponent id="IC3" name="microphone_icon" defaultContent="microphone.jpg" .../>
  <imageComponent id="IC4" name="mouse_icon" defaultContent="mouse.jpg" .../>
  <listBox id="LB1" name="List 1" isEnabled="true" currentValue="$x" ...>
    <item id="IT1" name="Item 1" defaultContent="Chris Hay" .../>
    <item id="IT2" name="Item 2" defaultContent="Lee Hardy" .../>
  </listBox>
  <imageComponent id="IC4" name="speaker_icon" defaultContent="speaker.jpg" .../>
</box>

<vocalForm id="VF1" name="Form 1" ...>
  <vocalPrompt id="VP1" name="Prompt 1" defaultContent="Please choose your favourite singer:
Chris Hay, Lee Hardy, ..." .../>
  <vocalInput id="V11" name="Input 1" currentValue="$y"/>
    <grammar id="GR1" name="Grammar 1" ...>
      <part id="P1" name="Part 1" structure="choice" ...>
        <item id="IT1" name="Item 1" defaultContent="Chris Hay" .../>
        <item id="IT2" name="Item 2" defaultContent="Lee Hardy" .../>
      </part>
    </grammar>
  </vocalInput>
  <vocalFeedback id="F1" name="Feedback 1" defaultContent="Your choice is $y" .../>
</vocalForm>

<synchronization>
  <source sourceId="V11"/>
  <target targetId="LB1"/>
</synchronization>
```

Appendix D. QOC representation of design space options in TEAM tool

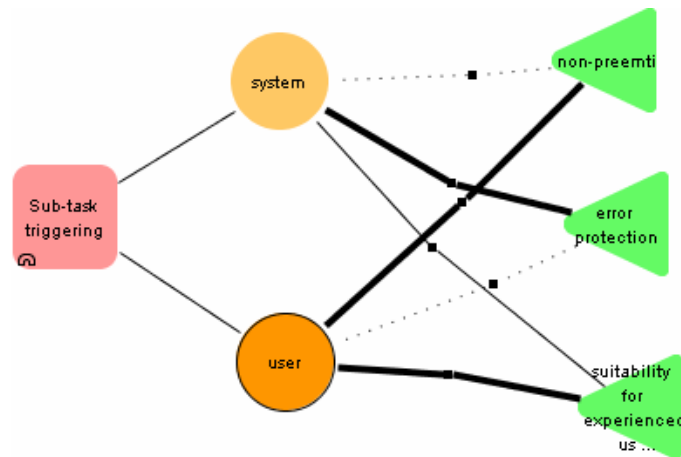


Figure D- 1 QOC representation of the *Sub-task triggering* design option

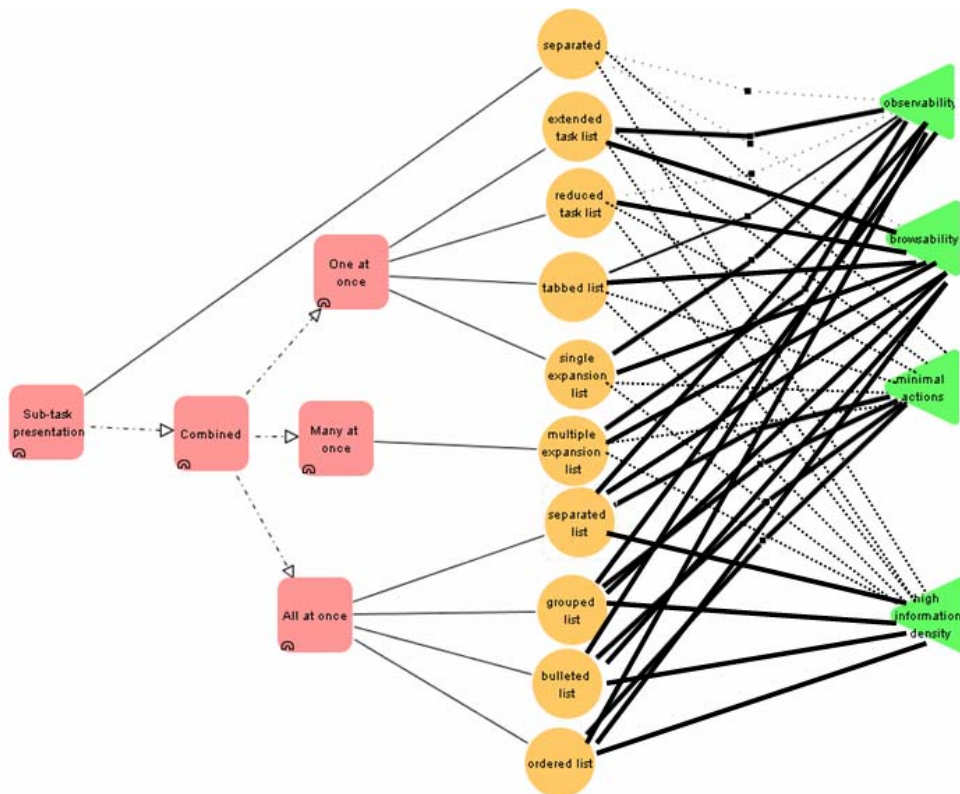


Figure D- 2 QOC representation of the *Sub-task presentation* design option

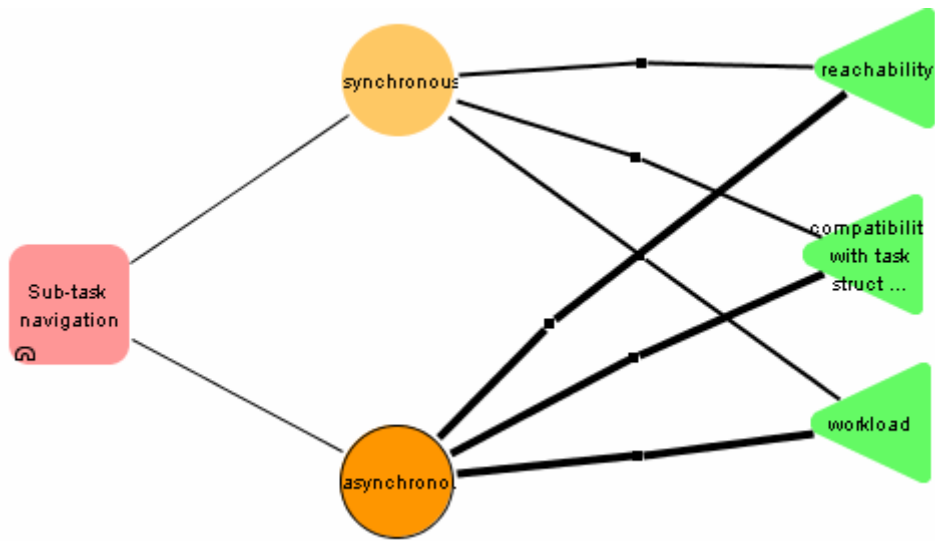


Figure D- 3 QOC representation of the *Sub-task navigation* design option

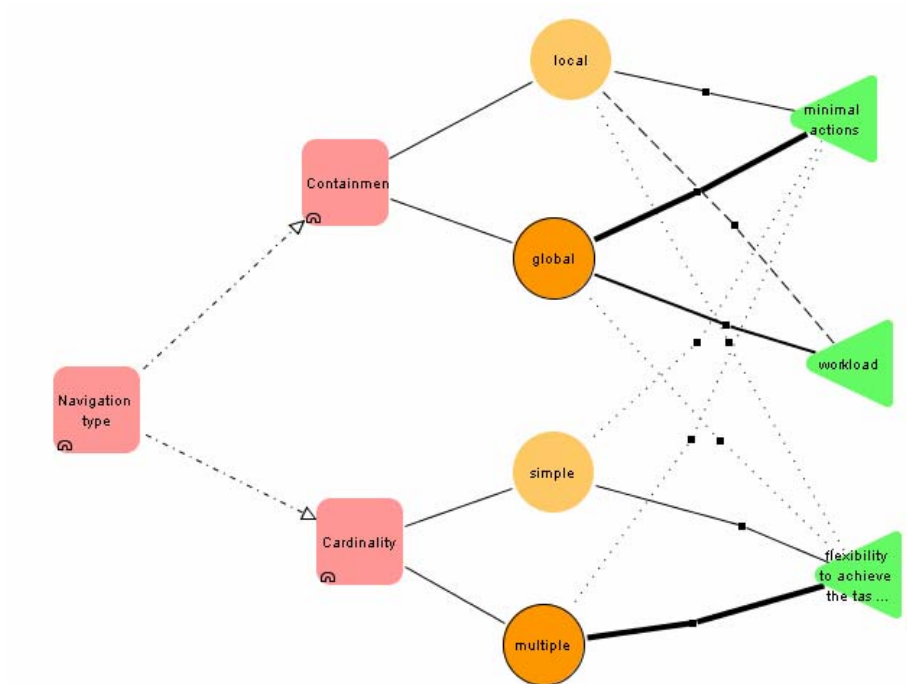


Figure D- 4 QOC representation of the *Navigation type* design option

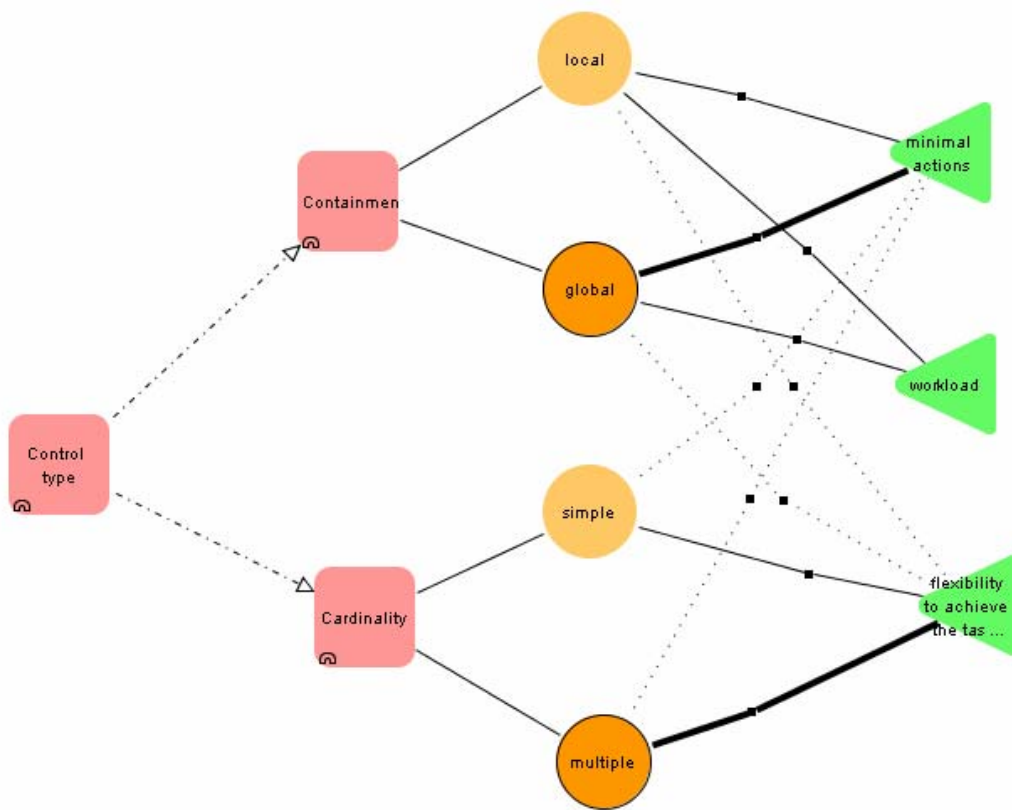


Figure D- 5 QOC representation of the *Control type* design option

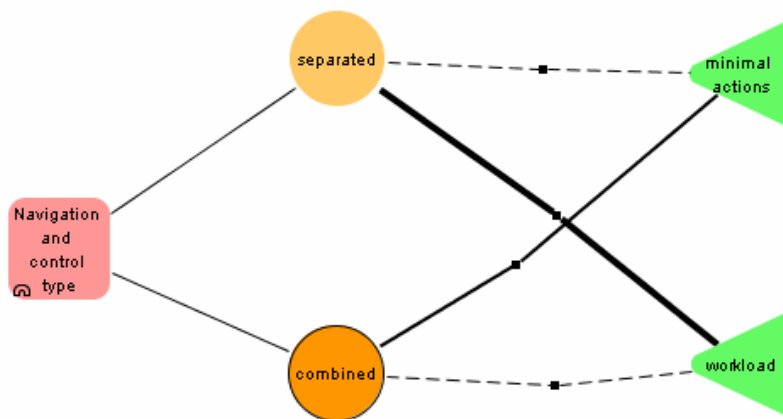


Figure D- 6 QOC representation of the *Navigation and control type* design option

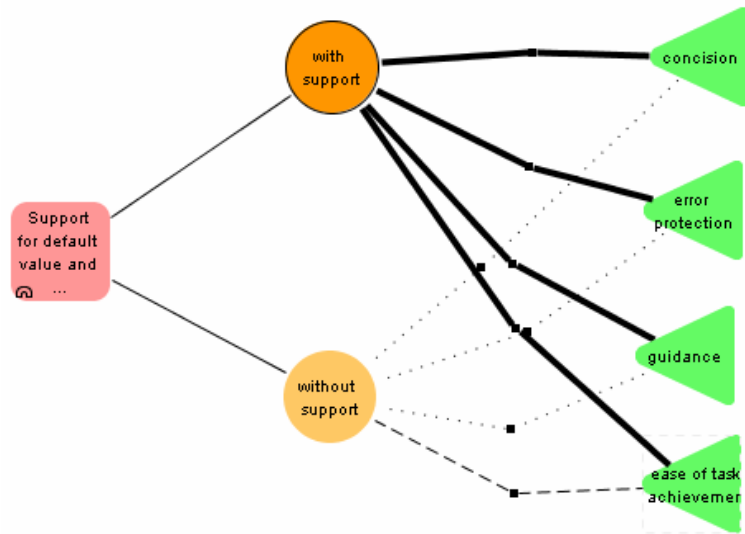


Figure D- 7 QOC representation of the *Support for default value and unit* design option

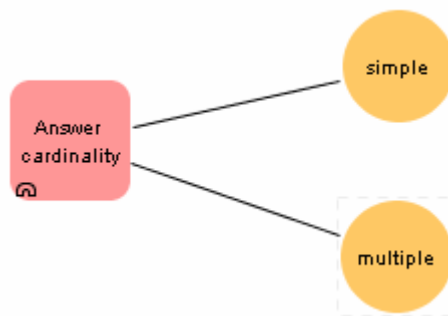


Figure D- 8 QOC representation of the *Answer cardinality* design option

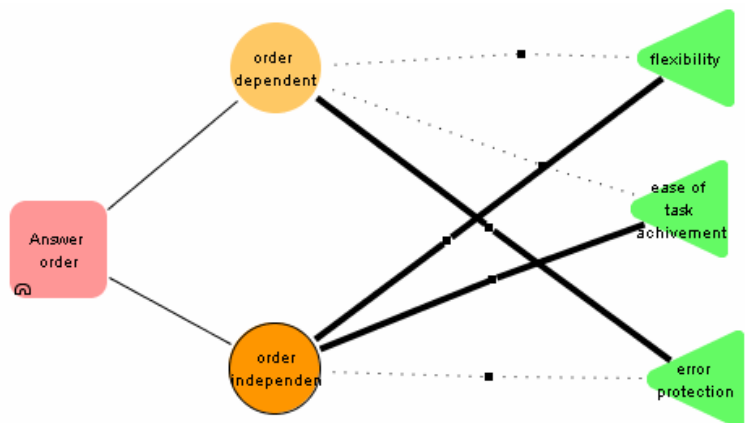


Figure D- 9 QOC representation of the *Answer order* design option

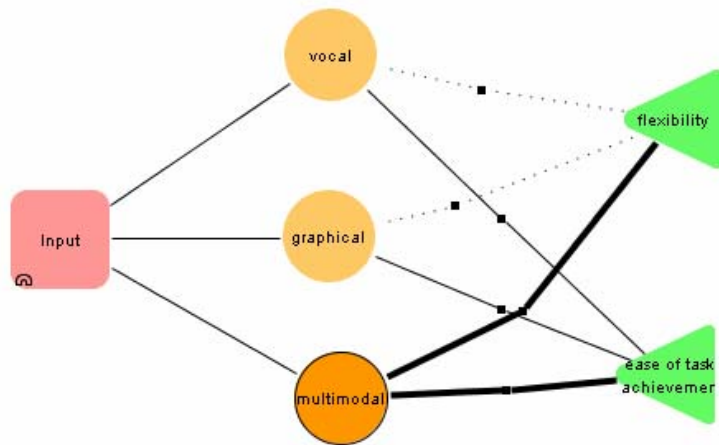


Figure D- 10 QOC representation of the *Input* design option

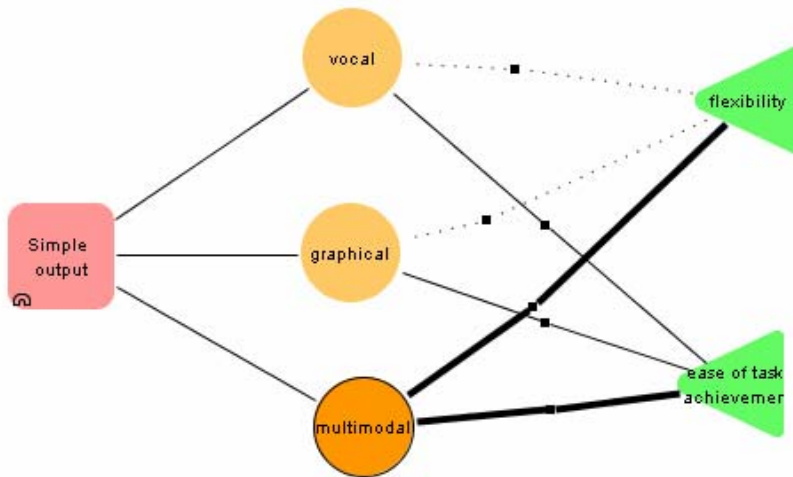


Figure D- 11 QOC representation of the *Simple output* design option

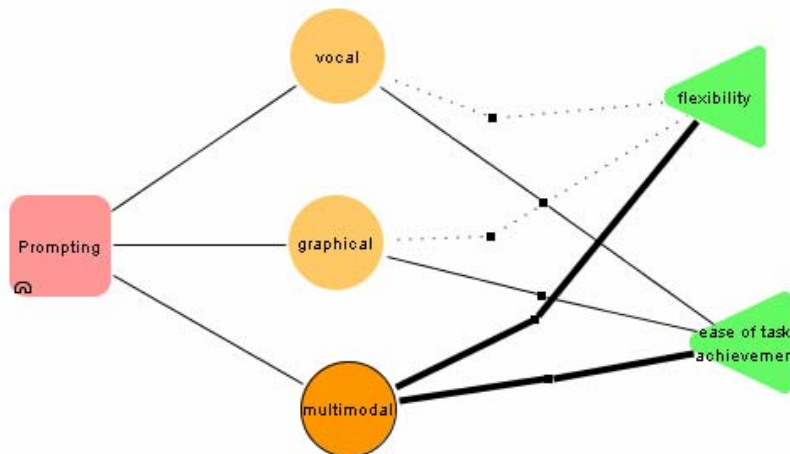


Figure D- 12 QOC representation of the *Prompting* design option

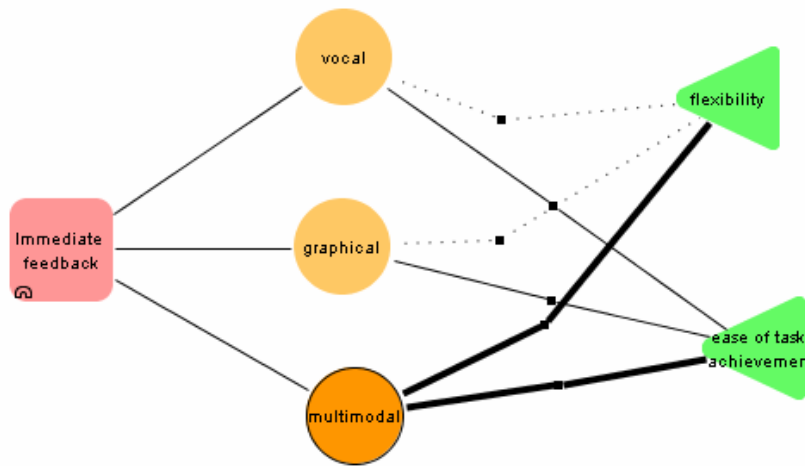


Figure D- 13 QOC representation of the *Immediate feedback* design option

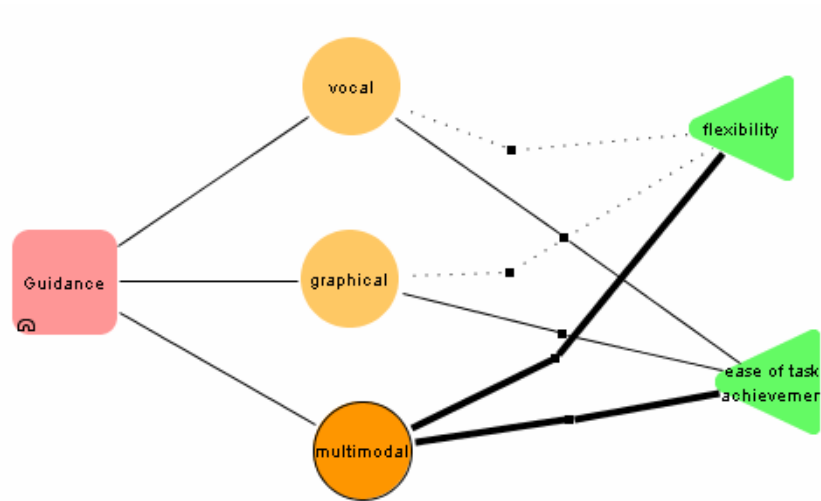


Figure D- 14 QOC representation of the *Guidance* design option

Appendix E. Acronyms

Acronym	Meaning
AC	Abstract Container
AGG	Attributed Graph Grammar
AIC	Abstract Individual Component
AIO	Abstract Interaction Object
AUI	Abstract User Interface
CARE	Complementarity, Assignment, Redundancy, Equivalence
CCXML	Call Control Markup Language
CIDL	Component Interface Description Language
CIO	Concrete Interaction Object
COCOMO	COConstructive Cost Model
CUI	Concrete User Interface
Db	Decibel
DISL	Dialog and Interface Specification Language
DTMF	Dual Tone Multi-Frequency
EMMA	Extensible MultiModal Addnotation Markup Language
FUI	Final User Interface
GC	Graphical Container
GIC	Graphical Individual Component
GUI	Graphical user Interface
HCI	Human-Computer Interaction
HTML	Hyper Text Markup Language
ICARE	Interaction CARE
IDE	Integrated Development Environment
IS	Information System
LHS	Left Hand Side
MB-IDE	Model Based Integrated Development Environment
MDA	Model-Driven Architecture
MDD	Model-Driven Development
M	Mean
MM	Multimodal
MONA	Mobile multimOdal Next generation Applications
MOST	Multimodal Output Specification Platform
NAC	Negative Application Condition
PC	Personal Computer
PDA	Personal Digital Assistant
PDCL	Pipeline Description and Configuration Language
QOC	Question, Option, Criteria
OMG	Object Management Group
RAD	Rapid Application Development
RHS	Right Hand Side
RUP	Rational Unified Process
SSML	Speech Synthesis Markup Language
UI	User Interface
USIXML	User Interface eXtensible Markup Language
UIDL	User Interface Description Language
TYCOON	Types of COOperationN
VC	Vocal Container
VIC	Vocal Individual Component
VUI	Vocal User Interface

Appendix E: Acronyms

XIML	eXtensible Interface Markup Language
XISL	eXtensible Interaction Language
VoiceXML	Voice eXtensible Markup Language
WIMP	Windows Icons Menu Pointers
WML	Wireless Markup Language