UNIVERSITE CATHOLIQUE DE LOUVAIN
SCHOOL OF MANAGEMENT
BELGIAN LABORATORY OF COMPUTER-HUMAN INTERACTION

A Transformational Approach for Developing Multimodal Web User Interfaces

By Adrian Stanciulescu

A dissertation submitted in fulfillment of the requirements for the degree of
Certificate of In-depth Studies in Management
Option "Information Systems"

Academic year 2005-2006

Table of Contents

Acknowledgement

I would like to express my thanks to:

- My advisor, Professor Jean Vanderdonckt, for his constant support and enthusiasm regarding my work

- My family and friends

- SIMILAR network of excellence (http://www. similar.cc), the European research task force creating human-machine interfaces similar to human-human communication of the European Sixth Framework Programme (FP6-2002-IST1-507609). This research is fully funded by SIMILAR.

- Didier Magotteaux and Vincent Wauters for the IBM Belgium grant received with the IBM Multimodal Toolkit™, Software Modeler™, and WebSphere™, with which this research is conducted.

Chapter 1 INTRODUCTION

1.1 Context

Today's graphical user interfaces (GUIs) do not let users communicate in ways that they naturally do with other human beings [Scot00]. Many end-users have limited literacy skills, typing skills, or use of their hands. The standard GUI does not work well for these users or for others in many situations: when users are moving around, using their hands or eyes for something else, or interacting with another person. To enjoy the benefits of ubiquitous computing, there is a need of newer, better interface paradigms. Multimodal user interfaces are one paradigm that can successfully address many of the aforementioned problems.

On the Internet, people use browsers to visit Web sites, access documents from networks, and fill out forms. With this growing capability to retrieve information, communications between users and their devices is receiving more attention. As devices become smaller, other means of input - in addition to keyboard or tap screen - are becoming necessary. Small handheld devices, including cell phones and PDA's, now contain sufficient processing power to handle multiple tasks. On some devices it is difficult to perform these tasks using only keyboard, stylus, or handwriting recognition. This has lead to a new application technology called *multimodal*, the use of multiple methods of communication between the user and a device. These methods include keypad, touch or tap screen, handwriting recognition, speech synthesis and voice recognition.

Multimodal user interfaces [Ovia99] represent a research-level paradigm shift away from conventional windows-icons-menus-pointers (WIMP) interfaces toward providing users with great expressive power, naturalness, flexibility and portability. 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 multimodal interface can accommodate a wide range of users, tasks, and environments - including users who are temporarily or permanently handicapped, usage in adverse settings (noisy environments, for example) or while mobile, and other cases for which any given single mode may not suffice. In many of these real-world instances, integrated multimodal systems have the potential to support entirely new capabilities that have not been supported at all by previous traditional systems.

Multimodal systems have been viewed as an attractive area for human computer interaction research since Bolt's seminal "Put That There" [Bolt80] for positioning objects on a large screen using speech and pointing. The promise of multimodal interaction has been and continues to be more natural and efficient human-computer interaction [Cohe98]. The multimodal design space is growing in popularity due to the increasing accuracy of perceptual input systems (e.g., voice recognition, handwriting recognition, vision recognition, etc.) and the increasing ubiquity of heterogeneous computing devices (e.g., cellular telephones, handheld devices, laptops, and whiteboard computers).

Multimodal applications can be developed successfully in the area of data services, such as:

- Accessing business information, support desks, order tracking, airline arrival and departure information, cinema and theater booking services, and home banking services.
- 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.

Far more people today have access to a computer with an Internet connection. Multimodal applications offer the promise of allowing everyone to access web based services from any online mobile device, making it practical to access the web anyplace, anywhere and anytime, whether at home, on the move, or at work. If the need for multimodal interaction extends to the network, then the Internet needs new technologies and standards to enable that functionality. Increasingly, Web developers are seeking ways to turn existing visually oriented web pages into multimodal ones, as multimodal interaction holds the promise of more natural dialogs with web-based services.

1.2 Motivation

The development of multimodal web user interfaces considered in the current dissertation is motivated by the following statements:

- *Modality interaction flexibility:* multimodal applications give users the flexibility to choose the interaction modality that is the most suitable for the considered task, as its achievement depends on: environment (e.g., noisy), context (e.g., driving in a car), complexity of task (e.g., directory assistance), device capability (e.g., small displays), preferences and disabilities of the user, (e.g. visually impaired).
- *Faster interaction:* the parallel input allows users to more quickly access and respond to information delivered by their devices.
- *Lower incidence of errors and easier error recovery:* being able to switch between modes of interaction (using a combination of keyboard, touch screen, stylus, telephone keys, and voice) decreases the occurrence of errors, because end-users can choose the mode most suited to different activities.
- *True device mobility:* the ability of switching between interaction modes (eyes-free, hands-free, audio-only) allows end-users to take full advantage of their mobile interaction devices.
- *Usability improvement:* multimodal UIs improve the usability of data services such as weather, driving directions, stock quotes, personal information management, and unified messaging by offering the possibility of developing a wide range of personalized and differentiated UIs.
- *Robust systems:* multimodal language is different then the monomodal ones, in the sense that they are simpler in many aspects, offering the possibility to build more robust systems [Ovia99]. This is due to the fact that multimodal languages are syntactically less complex, the fluency is higher, thus allowing a more doubtlessly debit.
- *Enhancement of device effectiveness:* as devices continue to get smaller, multimodal interaction can help increase the effectiveness of the device by combining multiple input and output modes of communication.
- *Improved experience:* multimodal applications improve the end-users experience with mobile devices and encourage the growth and acceptance of multimodal commerce on the web

- *Lack of multimodal applications:* although several real multimodal systems have been built, their development still remains a difficult task. Applications as well as development tools dedicated to the design of multimodal user interfaces are currently few and limited in scope.

1.3 Terminology

The goal of this section is to clarify the signification of the terms used in the field of multimodal interaction, such as: mode, modality, media and multimodality. The reason of clarifying this issue is due to the fact that the significations of these concepts have been a disputed subject as different authors propose different definitions. In the following we will present the definitions that will be used in the context of this dissertation.

1.3.1 Mode

The communication *"mode"* [Bell92] corresponds to the motor or sensorial system of the user. As described in [Schy05], the communication *"mode"* refers to the communication channel used by the two entities that interact. According to this point of view two available input modes corresponding to two motor systems of human beings are identified: the oral and the gesture modes. Moreover, corresponding to the five human being senses there are five available output modes: visual, auditive, tactile, olfactive and gustatory modes.

In this dissertation we consider an extended view over the above identified modes that cover in a more precise way our needs in defining the types of UI interactions. Thus, we consider four types of input communication modes, based on the implied sensorial system: graphical, vocal, tactile and gesture. As regarding output we have identified six communication modes, based on implied sensorial systems as well as on motor system: graphical, vocal, tactile, olfactory, gustatory and gesture (e.g., an avatar that uses his hands in order to express himself). 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.

1.3.2 Media

Most of the authors agree in defining *"media"* as a technical support for the information. In [Niga94] *"media"* is defined as a physical device that allows stocking or communicating the supported information. Consequently, the definition refers to all input devices (e.g., mouse, keyboard, microphone, etc.), to all output devices (e.g., screen, loud speakers, etc) as well as to the devices that allow stocking the information (e.g., CD Rom, DVD, etc) [Schy05]. As a conclusion, "*media"* is seen as being more then a *"physical device"* even if these two terms are used very often as synonyms.

1.3.3 Modality

Our view over the term *"modality"* is based on the definition given in [NIGA97a]. The interaction modality is seen as a couple of a physical device d and an interaction language L: $<d, L>$. A physical device is an artifact of the system that acquires (input devices) information (e.g., microphone, keyboard, mouse, etc.) or delivers (output device) information (e.g., screen, loud speakers). An interaction language defines a set of conventional symbols that convey meaning (e.g., restricted natural language, direct manipulation, unrestricted natural language). The symbols are generated by actions applied on physical devices. According to the definition given above some examples of modalities are defined in the following:

- *Speech input = (microphone, restricted vocabulary-oriented natural language)*: as the employed interaction type is vocal, the modality is said to be vocal
- *Written natural language = (keyboard, command language):* as the employed interaction type is graphical, the modality is said to be graphical
- *Graphical input = (mouse, direct manipulation):* as the employed interaction type is graphical, the modality is said to be graphical
- *Graphical output = (screen, tables):* as the employed interaction type is graphical, the modality is said to be graphical
- *Vocal output = (loud speakers, unrestricted natural language):* as the employed interaction type is vocal, the modality is said to be vocal

1.3.4 Multimodality vs. Multimedia

A multimodal system is described in general as a system that supports communication with the user through different modalities. The term *"multi"* implies the use of more then one modality. Multimodality refers to output as well as to input modalities:

- Input multimodal systems are employing at least two different input modalities
- Output multimodal systems are employing at least two different output modalities.

In this dissertation the definition of multimodality is based on a system-centered view. Thus, a multimodal system is a system that has the capacity to communicate with a user through different types of communication modes and to extract and convey meaning automatically [Niga97c]. But, multimedia systems are also using multiple types of communication modes. Consequently, a question comes into sight: what is the difference between a multimodal system and a multimedia system? The answer to this question is given in [Cout92]. A multimedia system allows the acquisition, the stocking and the distribution of data, while a multimodal system is capable of acquiring, interpreting data and stocking these interpretations as well as distributing them. In conclusion, a multimodal system is a system with multimedia capabilities which offers in the same time the possibility of semantic treatment of data.

1.4 Thesis statement

The scope of the current thesis is outlined by the following statements:

- We focus on the development of multimodal web user interfaces of information systems. *Multimodal web UIs* are user interfaces that give end-users the flexibility to choose the interaction modality that is the most suitable for the task at the given moment when manipulating online devices. With respect to the interaction modality we consider three types of UIs:
  - ✓ Graphical UI: the modality employed by the user to interact with the system is entirely graphical (monomodal UI)
  - ✓ Vocal UI: the modality employed by the user to interact with the system is entirely vocal (monomodal UI)
  - ✓ Multimodal UI: the graphical and the vocal modalities are employed in the interaction between the user and the system.

- We consider the conceptual and methodological aspects for developing multimodal web UIs based on a model-to-model transformational approach.
- We introduce in the development life cycle towards a final UI, a design space that will ease the development process in a structured way in terms of design options, thus requiring less design effort from the part of designers.
- We target this dissertation to all research organizations that dedicate their work to the methodological development of user interfaces for information systems.

1.5 Reading map

This dissertation is structured in 6 chapters as follows:

Chapter 1 consists of an introduction in the multimodal interaction research area, specifying the motivation of this dissertation and the terminology used in the literature.

Chapter 2 presents the state of the art in the field of multimodal interaction. First, a description of 3 conceptual multimodal frameworks and a comparison between them is provided. Further, the features of a set of 8 monomodal/multimodal languages are presented. The chapter also identifies the characteristics of a series of representative multimodal user interface development tools. We conclude with a summary of the state of the art. Based on this summary we establish a list of requirements defined along with the corresponding motivations.

Chapter 3 introduces the concepts of our framework for the development of multimodal web applications. The user interface description language selected for this purpose is described along with its extended syntax, semantics and stylistics.

Chapter 4 presents our transformational method for producing multimodal web applications. First, the details concerning the specification of transformations are described. Further, a design space composed of design options for graphical and multimodal web user interfaces is introduced. Based on these design options the 4 steps of the transformational approach are detailed. Finally, we present the tool support for the transformational method.

Chapter 5 illustrates the transformational approach for the development of multimodal web user interfaces based on design options for two case studies. The first one concerns the development of an on-line polling system. The second one concerns the development of a car rental system.

Chapter 6 concludes this dissertation by differentiating between the stable knowledge and the knowledge that was acquired, but has to be improved and assessed. Finally the future works are proposed.

Chapter 2 STATE OF THE ART

Chapter 2 presents the state of the art in the field of multimodal interaction. Section 2.1 provides a description of 3 conceptual multimodal frameworks and a comparison between them. Section 2.2 presents the features of a set of 8 monomodal/multimodal languages. The characteristics of a series of representative multimodal user interface development tools are identified in Section 2.3. Section 2.4 concludes with a summary of the state of the art.

2.1 Conceptual frameworks for multimodal systems

2.1.1 TYCOON framework

Tycoon (Types of COOperatioN) [MART01] is a framework for observing, evaluating and specifying cooperation among modalities during multimodal human-computer interaction. 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:

- **Transfer.** The transfer cooperation type of several modalities specify that a chunk of information produced by a modality is used by another modality. The transfer can appear either between two input modalities, or between two output modalities, or between an input modality and an output modality. The goals of transfer cooperation type are:
  - ✓ Translation: for instance, in hypermedia interfaces a mouse click provokes 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 (here, that,…)
  - ✓ Enable a faster interaction: when a part of a uttered sentences has been misrecognized, it can be edited using a keyboard so that the user doesn't have to type or to utter again the hole sentence.
- **Equivalence.** The equivalence cooperation type of several modalities means that a chunk of information may be processed as an alternative, by either of the modalities. The goals of equivalence cooperation type are:
  - ✓ Improve recognition command: the user of a graphical editor may specify a command either through speech or through the selection of a button with a pen, so as when the speech recognizer is not working accurately (e.g., because of noise), the user can select the command with the pen
  - ✓ Adaptation to the user by customization: the user is allowed to select the modality he prefers
  - ✓ Faster interaction: because it allows the system or the user to select the fastest modality.
- **Specialization.** Modalities that cooperate by specialization precise that a specific kind of information is always processed by the same modality. The goals of specialization cooperation type are:

  - ✓ Interpretation: the user is helped to interpret the events produced by the computer
  - ✓ 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.
- **Redundancy.** Several modalities that cooperate by redundancy are processing the same information (e.g., if the user types "quit" with the keyboard and utters "quit", this redundancy can be used by the system to avoid a confirmation dialog, thus enabling a faster interaction). Two observations have been made:
  - ✓ Regarding intuitiveness: a case study reveal that sometimes the users select their options (e.g., the town) both by speech and touch of tactile screen
  - ✓ Regarding learnability: a redundant multimodal output involving both visual display of a text and speech utterance of the same text enables a faster graphical interface learning.
- **Complementarity.** Complementarity considers several modalities that are processing each one different chunks of information which are merged afterwards. The goals of complementarity cooperation type are:
  - Faster interaction: because the two modalities can be used simultaneously and they convey shorter messages which are also better recognized than longer messages
  - Improve interpretation: for an expert the graphical output is sufficient, but for novice users a textual output is needed too.

COMIT is a TYCOON framework tool-based that offers to the users a multimodal interface which allows them to build graphical interfaces. It features several types and goals of cooperation between speech recognition, a keyboard and a mouse. COMIT is defined by a language command which is used to specify the cooperation between modalities.

2.1.2 CARE properties

The CARE properties (Complementarity, Assignment, Redundancy and Equivalence) represent a way of characterizing the relationships that can occur between different interaction modalities available in multimodal user interfaces. A modality is described as a couple of a physical device *d* and an interaction language *L*: <*d*, *L*> (see Section 1.3.3). In order to give a formal definition of the CARE properties some notions 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 notions defined above, the following formal definitions of the CARE properties are specified:

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

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

  We consider the following:

  Modalities:

  m1=speech input <microphone, restricted vocabulary-oriented natural language>
  m2=written natural language <keyboard, command language>

  State s = a multimodal user interface with an unfilled textfield widget
  State s' =a multimodal user interface in which the textfield widget from state s is filled
  Example: a text field can be fulfilled by an agent using any of the modalities *m1* or *m2*.

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

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

  We consider the following:

  Modality:

  m = written natural language <keyboard, command language>

  State s = a multimodal user interface with an unfilled textfield widget
  State s' =a multimodal user interface in which the textfield widget from state s is filled.
  Example: a text field can be fulfilled by an agent using only the modality *m*. No other modality is used to reach *state s'*.

- **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*:

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

  We consider the following:

  Modalities:

  m1= speech input <microphone, restricted vocabulary-oriented natural language>
  m2 = graphic input <mouse, direct manipulation>

  State s = a multimodal user interface with an unfilled combobox widget
  State s' = a multimodal user interface in which the combobox widget from state s is filled.
  Example: a combo box can be fulfilled by an agent either by using modalities *m1* and *m2* in parallel, either 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 to be treated as if they were parallel).

- **Complementarity (C).** Modalities of a set *M* must be 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):

Complementarity (s, M, s', tw) $\Leftrightarrow$ *(Card(M) >1)* $\wedge$ *(Duration(tw)* $\neq \infty$ *)* $\wedge$
($\forall$ *M'* $\in$ ***P****M (M'*$\neq$ *M* $\Rightarrow \neg$ *REACH (s, M', s')))* $\wedge$ *REACH (s, M, s')* $\wedge$
(Sequential (M, tw) $\vee$ *Parallel (M, tw)).*

We consider the following:

Modalities:

m1= speech input <microphone, restricted vocabulary-oriented natural language>
m2 = written natural language: <keyboard, command language>

State s = a multimodal user interface with an unfilled textfield widget allowing to input the name

State s' = a multimodal user interface in which the textfield widget from state s is filled.

Example: modality m1 is employed by the user to utter the first part of his/her name, while *m2* is used to complete the fulfillment of the task. None of the modalities taken individually can not be used to reach *state s'*.

ICARE platform (Interaction CARE – Complementarity, Assignment, Redundancy and Equivalence) is a component-based approach which allows an easy and rapid design and development of multimodal user interfaces [BOUC04a]. There are two types of components: elementary components, which are building blocks used to describe pure modalities (i.e., device components and interaction language components) and composition components which describe the combined usage of modalities (i.e., the CARE properties). The platform enables the designer to graphically manipulate and assemble ICARE components in order to specify the multimodal interaction dedicated to a given task of the interactive system. From this specification, the code is automatically generated. Some multimodal systems were developed using ICARE Platform:

- MEMO PDA: allows users to annotate physical locations with digital notes which have a physical location and are then read / removed by other mobile users.
- FACET: is a simulator of Rafale (a French military plane). One prototype has been realized with ICARE Components for few tasks.

2.1.3 W3C Multimodal Interaction Framework

The purpose of the W3C Multimodal Interaction Framework [Lars03b] is to identify:

- Basic components for multimodal systems
- Markup languages used to describe information required by components (W3C markup languages)
- 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 1-1 illustrates the basic components of the W3C Multimodal Interaction Framework:

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

- 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 which may be used for coordinated 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 multimodal 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, does the display supports color or not).

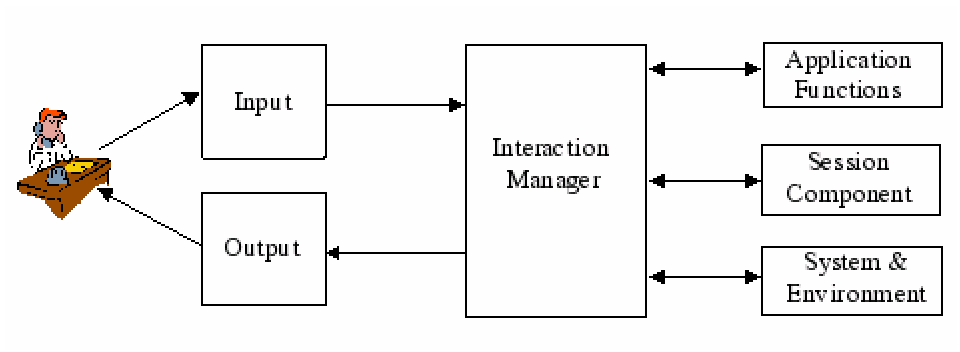


Figure 1-1. W3C Multimodal Interaction Framework

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

1. *Sequential multimodal input:* a sequential input is one received on a single modality. The modality may change over time. Sequential input implies that it must be possible to specify what modality or device to use for input in sequential multimodality and hint or enforce modality switches.
2. *Simultaneous multimodal input:* simultaneous multimodal inputs imply 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:* composite input is the input received on multiple modalities at the same time and treated as a single, integrated compound input by downstream processes.

2.1.4 Comparison of conceptual frameworks

A first difference between the frameworks results from the way they are defining the notion of modality. While in the TYCOON framework a modality is defined as a process which analysis and produces chunks of information, in the case of CARE properties a modality is a couple of a physical device d with an interaction language L : <d, L>. The W3C Multimodal Interaction 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 framework, concept that is missing in the case of CARE properties. More then that, due to the fact that in several existing systems sounds are somehow specialized in notification errors (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

If with CARE properties it can be defined the relationships between [Niga97b] devices and interaction languages, interaction languages and tasks, or between different modalities, in TYCOON framework the properties are used in a more restrictive way as they are describing only various types of cooperation between modalities. Another difference comes from the point of view of treating the interaction between the system and the user. With CARE it is possible to define cooperation between different modalities from the system point of view (*system CARE properties*) as well as from the user's point of view (*user CARE properties*). The *user CARE properties* refer to the user's preferences that affect his/her choice for input modalities. With TYCOON only the system point of view is considered.

Some similarities can be found between TYCOON and CARE properties on one side and W3C multimodal interaction requirements, on the other side. The *Redundancy* property from TYCOON and CARE frameworks could be expressed by using modalities sequentially or in parallel, which corresponds respectively to sequential and simultaneous multimodal input requirements identified by W3C Multimodal Interaction Working Group. In the same way, the *Complementarity* property supposes both a sequential or parallel use of modalities treated as a single which corresponds to sequential and composite multimodal input requirements, respectively.

2.2 Current (mono/multi)modal languages

2.2.1 XISL

XISL (eXtensible Interaction Sheet Language or eXtensible Interaction Scenario Language) [Kats03] is a XML language for developing web-based multimodal applications. The application consists of interaction scenarios between the user and the system. In principle, a scenario is composed of a sequence of exchanges that contains a set of user's multimodal inputs and the system's actions corresponding to the inputs.

One of the main features of XISL is the separation of the content from the interaction. The content is held into XML/HTML files while the interaction scenario is described separately into XISL documents. This creates some advantages of XISL against other multimodal languages:

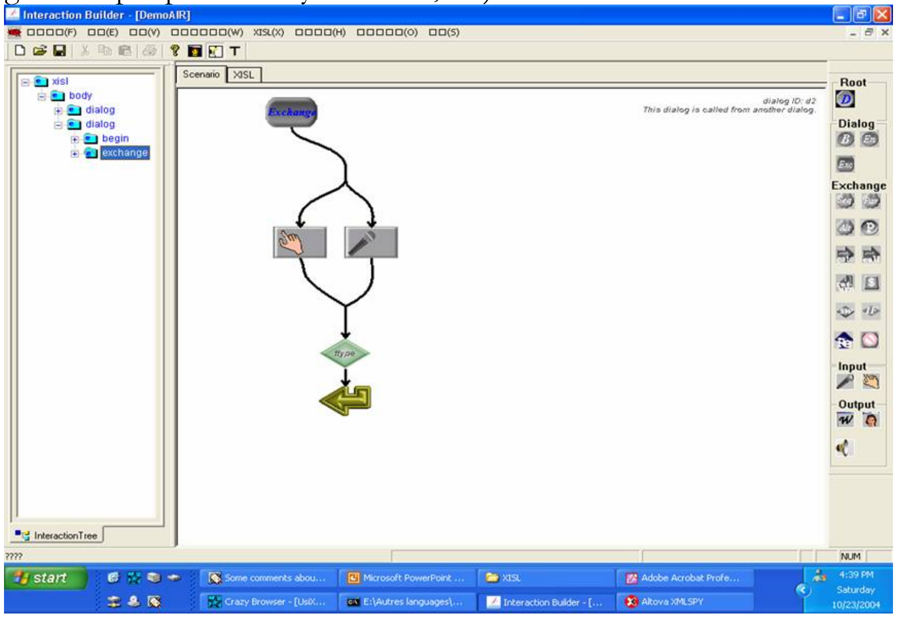
- System developers can reuse XML/HTML files as well as the XISL files
- Improved readability is offered

Applications developed in XISL allow multiple types of interaction depending on entity that has the initiative: user initiative, system initiative or even mixed initiative (i.e., the user and the system apportion their initiative). The language was developed to be supported by different types of devices like: PCs, mobile phones, PDAs and it offers the possibility of easily extending them. This flexibility is given by the use of non strict attribute values for XML elements used to specify the input/output of the user/system. The input/output can be used cooperatively as follows: parallel input/output, sequential input/output, alternative input.

The goal of XISL is to provide a common language for web-based multimodal interaction that satisfies three main features:

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

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 graphic user interface design for domain-specific prototyping (Figure 1-2). The interaction scenario is presented under the form of a state transition diagram. Nodes of the diagram or multimodal 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, etc).



dialog ID: d2
This dialog is called from another dialog.

Figure 1-2. The Interaction Builder graphical user interface

2.2.2 XIML

XIML stands for eXtensible Interface Markup Language [Puer02a]. It is a XML-based language whose main goal is to enable a framework for the definition and interrelation of interaction data. Interaction data refers to the data that defines and relate all relevant elements of a user interface. From the structure point of view XIML language includes the following representational units:

- Components: XIML is an organized collection of interface elements that are categorized in major interface components found in interface models:
    - ✓ User tasks: defines a hierarchical decomposition of tasks and subtasks, and the expected flow between those tasks
    - ✓ 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: is a hierarchy of interaction elements made of concrete objects which communicate with users
    - ✓ Dialog elements: structured collection of elements that determine the interaction actions available to the users
- Relations: definition or statement than links two or more XIML elements inside of the same component or between different components
- Attributes: features or properties of elements

One of the important uses of XIML can be in the development of user interfaces that must be displayed in a variety of devices. XIML 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 XIML makes between the definition of a user interface and the rendering of that interface - the actual display of the interface on a target device. In the XIML framework, the definition of the interface is the actual XIML specification and the rendering of the interface is left up to the target device to handle. There are a number of converters [Puer02b] used to transform a XIML specification 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.

2.2.3 UIML language

UIML [Abra04] is an XML-based language that provides a device-independent method to describe a user interface. UIML is also independent of any user interface metaphor, such as graphical user interface or vocal user interface.

UIML allows describing three aspects: the appearance of the user interface, the user interaction with the user interface and the connection of the user interface with the application logic. There are four key concepts that underlie UIML:

1. **UIML is a meta-language.** UIML defines a small set of powerful tags (e.g., tags to describe a part of a UI, tags to describe a property of a UI part) that are independent of any UI metaphor (e.g., graphical UI, vocal UI), target platform (e.g., PC, phone), or target language to which UIML will be mapped (e.g., Java, HTML, VoiceXML). In order to use UIML a toolkit vocabulary must be added. The vocabulary specifies a set of classes of parts, and properties of the classes. Different groups of people can define different vocabularies, depending on their needs. 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), while another group might define a vocabulary whose classes match abstractions used by a UI designer.
2. **UIML separates the elements of a UI.** The separation in UIML identifies:

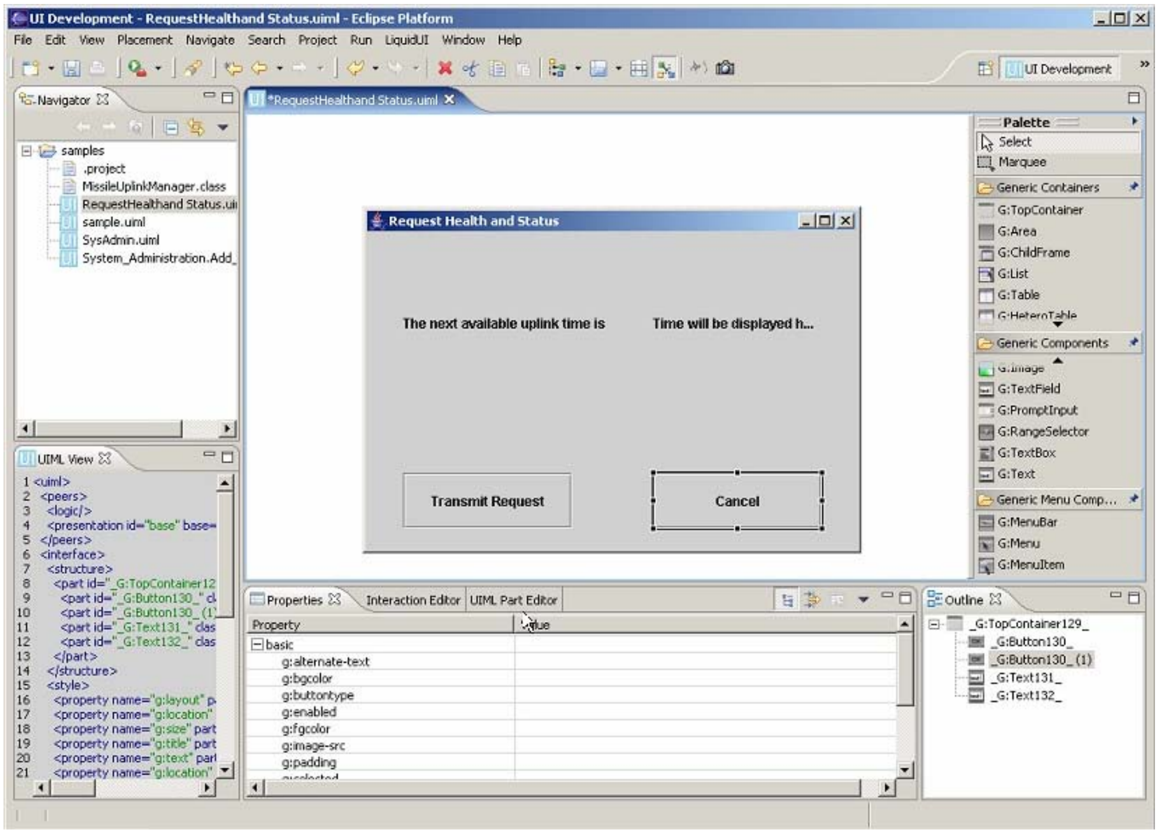
- ✓ What parts comprise a UI, the presentation style for each part
- ✓ The content of each part (e.g., text, sounds, images) and binding of content to external resources (e.g., XML resources, or method calls in external objects)
- ✓ The behavior of parts when a user interacts with the interface as a set of rules with conditions and actions
- ✓ The connection of the UI with the outside world (e.g., to business logic)
- ✓ The definition of the vocabulary of part classes.

3. **UIML views the structure of a UI, logically, as 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 (<structure>) and to dynamically modify the structure (<restructure>).
4. **UIML allows UI parts and part-trees to be packaged in templates.** Templates may then be reused in various interface design. This is a missing feature of other XML languages, such as HTML and WML.

Thanks to its features, UIML is particularly useful for creating multiplatform and multimodal UIs. To create multiplatform UIs, concept 1 is used to create a vocabulary of part classes (e.g., defining a class *Button*) and the concept 2 is used to separately define the vocabulary by specifying a mapping of the classes to target languages (e.g., mapping UIML part class *Button* to class *java.awt.Button* for Java and to the tag <*button*> for HTML 4.0).

To create multimodal UIs, a multiplatform UI should be created and then annotate each part 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 *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.

UIML Development Tool (Figure 1-3) allows user interface designers to generate high fidelity interfaces and production code. The tool is a plug-in for the Eclipse IDE. UIML is also supported by LiquidUI, a tool that integrates a set of converters for different software platforms (e.g., Java, HTML, WML, VoiceXML, C++, etc.). The disadvantage of converters is that if the target language changes from the specification point of view, a modification of the existing converter should be considered or a new converter has to be implemented.



```
<uiml>
  <peers>
    <logic/>
    <presentation id="base" base=
  </peers>
  <interface>
    <structure>
      <part id="_G:TopContainer12
        <part id="_G:Button130_" cl
        <part id="_G:Button130_(1)
        <part id="_G:Text131_" clas
        <part id="_G:Text132_" clas
      </part>
    </structure>
    <style>
      <property name="g:layout" p
      <property name="g:location"
      <property name="g:size" part
      <property name="g:title" part
      <property name="g:text" part
      <property name="g:location"
```

Figure 1-3. UIML Development Tool

2.2.4 DISL

DISL (Dialog and Interface Specification Language) [Bleu04] is a XML language defined in the context of the W3C Multimodal Interaction Framework (see Section 2.1.3).

DISL is based on a UIML subset (see Section 2.2.3), which is extended by rule-based description of state-oriented dialogs for the specification of advanced multimodal interaction and the corresponding interfaces. Comparing with UIML, DISL contains several modified or new constructs that increase the flexibility of the language. For example, the use of a <widget> element with a "generic-widget" attribute instead of a <part> element with a "class" attribute in order to ensure that the description is generic. Due to this improvement, DISL becomes suited for the creation of multimodal interfaces. The use of "generic-widget" attribute provides a classification for the type of object the given <widget> is (e.g., command, variable field, text field, etc.). A DISL renderer can then use this classification to create interface components appropriate to the interaction mode in which a given generic-widget will operate.

DISL is designed for mobile devices with limited resources. The current implementation of DISL is made on mobile phones that control the playback of MP3 files on a PC.

2.2.5 VoiceXML

The scope of VoiceXML [W3C04b] is to provide a standard dialog design language that could be used by developers to build web-based vocal applications. VoiceXML is a markup language that supports web-based vocal user interface development and minimizes client/server interactions by specifying multiple interactions per document. Voice XML is easy to use for simple interactions, and yet provides language features to support complex dialogs. The language describes the human-machine interaction provided by voice response system, which includes:

- Output of synthesized speech (text-to-speech)
- Output of audio files
- Recognition of spoken input
- Recognition of DTMF input
- Recording of spoken input
- Telephony features such as call transfer and disconnect

VoiceXML is supported by a series of tools between which IBM WebSphere Voice Toolkit offers one of the most complete set of features necessary to deploy VoiceXML-based application. The WebSphere Voice Toolkit is powered by Eclipse technology and makes it easy to develop VoiceXML applications without having to know the internals of voice technology. It offers a full-featured voice development environment including:

- Graphical communication flow builder (Figure 1-4)
- VoiceXML development and debugging
- Grammar development and debugging
- Pronunciation builder
- Call Control extensible Markup Language (CCXML) development.

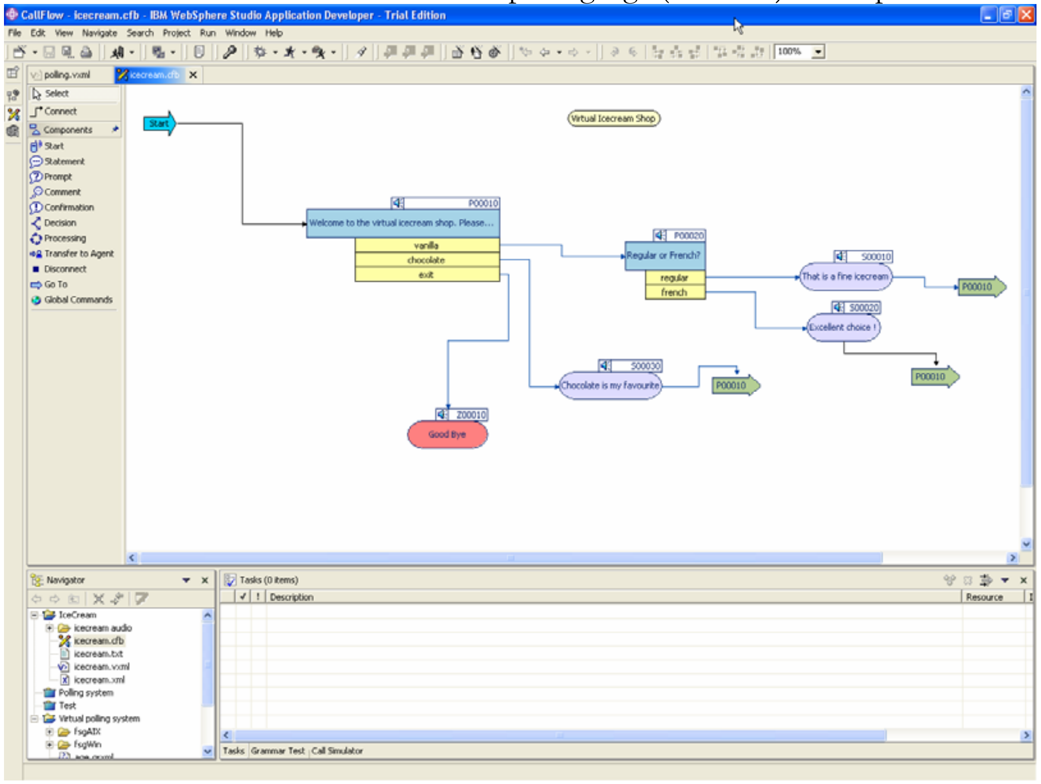


Figure 1-4. IBM WebSphere Voice Toolkit – communication flow builder perspective

2.2.6 XHTML+Voice (X+V)

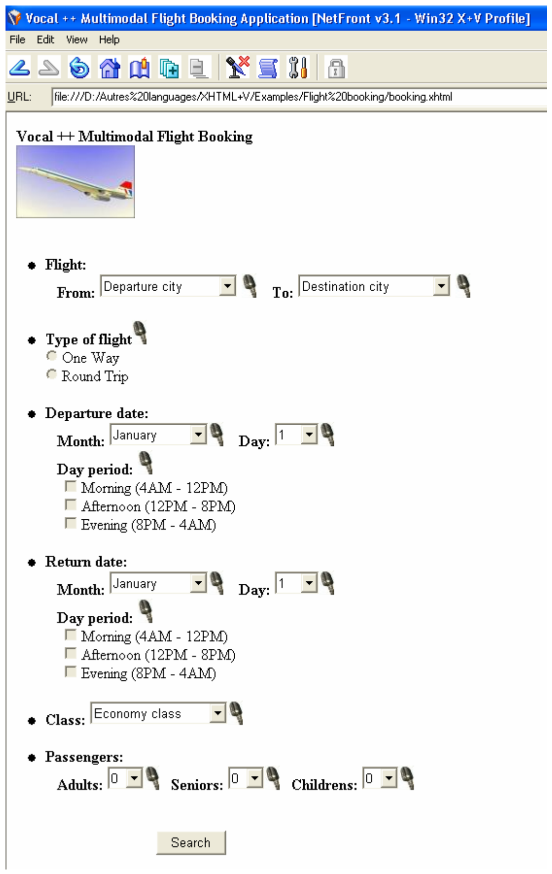
XHTML+Voice [W3C04b], or X+V for short, is a markup language for developing multimodal user interfaces for the web. Web developers can create web pages that allow end-users use voice input and output as well as traditional graphical interaction. X+V achieves this objective by providing a simple way to add voice markup to XHTML.

X+V is based on XHTML, VoiceXML and XML events. For graphical interaction, X+V uses XHTML standard. For vocal interaction it uses a simplified subset of VoiceXML standard. For correlating the vocal elements with the graphical ones X+V uses XML events standard.

X+V specification is interpreted by multimodal web browsers. Graphical elements of the language specify how a user interface looks like and how it should behave when the user types, points or clicks. Similarly, vocal elements specify how the multimodal system should behave when the end-user speaks to it and which are the vocal responses provided by the system. Each component of a multimodal interface built in X+V, the graphical and the vocal one, are treated by the multimodal browser using a different engine (i.e., a graphical engine and a speech engine, respectively)

Multimodal web applications built on X+V can be accessed by voice devices, browser-based devices and new multimodal devices. Consequently, the user can interact with the application using a keyboard, a stylus or by voice, depending on the environmental context (noisy environment or not, possibility of using her hands at a given moment, etc.). Thus, X+V multimodal applications give users the flexibility to choose the mode of interaction that is the most suitable for the task at the given moment. This may include the use of multiple modes of communication to give an enhanced user experience. For example the user can alternate between speaking and typing in the interface. Regarding the CARE properties presented in Section 2.1.2, X+V can afford *Assignment, Equivalence* and *Redundancy* properties.

X+V application can be developed and debugged using the IBM Multimodal Toolkit integrated into IBM Rational Web Developer. For interpreting the X+V application two multimodal browsers are available: Opera browser and NetFront browser (Figure 1-5).



Vocal ++ Multimodal Flight Booking Application [NetFront v3.1 - Win32 X+V Profile]

File Edit View Help

URL: file:///D:/Autres%20languages/XHTML+V/Examples/Flight%20booking/booking.xhtml

Vocal ++ Multimodal Flight Booking

- **Flight:**
  From: Departure city **To:** Destination city
- **Type of flight**
  - One Way
  - Round Trip
- **Departure date:**
  Month: January **Day:** 1
  Day period:
  - Morning (4AM - 12PM)
  - Afternoon (12PM - 8PM)
  - Evening (8PM - 4AM)
- **Return date:**
  Month: January **Day:** 1
  Day period:
  - Morning (4AM - 12PM)
  - Afternoon (12PM - 8PM)
  - Evening (8PM - 4AM)
- **Class:** Economy class
- **Passengers:**
  Adults: 0 **Seniors:** 0 **Childrens:** 0

Search

Figure 1-5. NetFront multimodal browser

2.2.7 TeresaXML

TeresaXML is an XML-based language that addresses the design of multiplatform user interface [Mori04]. The model-based approach employed by TeresaXML is composed of a number of steps that allows designers to start with an envisioned task model and then to derive concrete and final user interfaces for multiple devices (Figure 1-6).

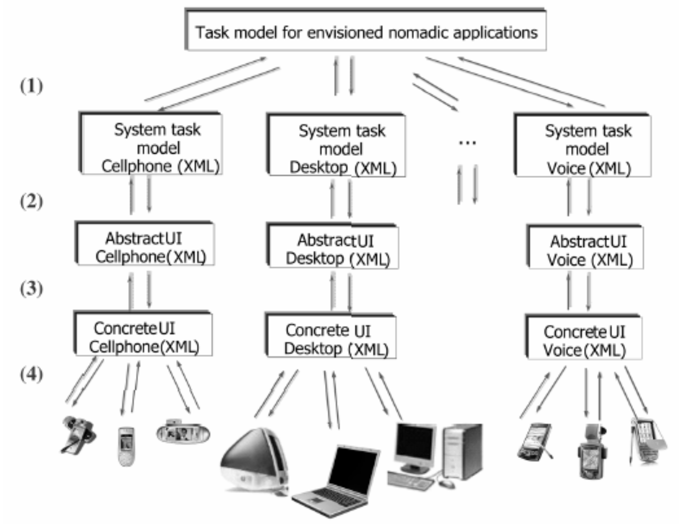

Task model for envisioned nomadic applications
(1)
System task model Cellphone (XML)
System task model Desktop (XML)
...
System task model Voice (XML)
(2)
Abstract UI Cellphone (XML)
Abstract UI Desktop (XML)
Abstract UI Voice (XML)
(3)
Concrete UI Cellphone (XML)
Concrete UI Desktop (XML)
Concrete UI Voice (XML)
(4)

Figure 1-6. Derivation of multiplatform user interfaces from an envisioned task model

In the first phase designers develop a single task model which addresses the possible context of use and the various platforms involved, including a domain model aiming to identify all the objects that have to be manipulated to perform tasks and the relations among such objects.

The next phase (see step 1 in Figure 1-6) consists of developing a system task model for each different platforms considered. The designers have to filter the task model according to the target platform, thus obtaining various platform-dependent task models. In step 2 designers will derive abstract user interfaces for each system task model. The result of this phase will be an abstract description of the user interface composed of a set of abstract presentations that are identified through an analysis of the task relations. Each presentation will be specified by means of abstract interaction objects composed through various operators (grouping, ordering, hierarchy, relation). The next step produces a concrete user interface that is completely platform-dependent and has to consider the specific properties of the target platform. Step 4 represents the code generation phase. The code for the target environment is generated automatically from the concrete user interface. The design of multimodal web UIs implies the use of several design options and their associated values.

In order to provide tool support for this approach, XML languages have been defined for task model, abstract model and concrete model. TeresaXML language is supported by Teresa tool, a transformation-based environment that addresses the design of interactive applications at different abstraction levels for various types of platforms (e.g., desktop, PDAs, cell phones). The main transformations supported by Teresa are those described in the above phases. All transformations are hard coded, embedded, and unique. The final objective of the tool is to produce multimodal (graphical and vocal) user interfaces through the generation of XHTML+Voice code.

2.2.8 EMMA

EMMA (Extensible MultiModal Addnotation markup language) [W3C04a] is a markup language used to represent information automatically extracted from the input of users which manipulate multimodal user interfaces. The language is capable to convey meaning for different types of input: text, speech, handwriting and combinations of any previous modalities.

EMMA is considering the following types of input (see Section 2.1.3):

- Single modality input
- Sequential modality input, that is: single-modality inputs presented in sequence
- Simultaneous modality input: imply that the inputs from several modalities are interpreted one after another in the order that they where received instead of being combined before interpretation
- Composite modality input: input received on multiple modalities at the same time and treated as a single, integrated compound input by downstream processes

The language will be used primarily as a standard data interchange format between components of a multimodal system. EMMA will 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.

2.3 Multimodal user interface development tools

2.3.1 MONA

MONA (Mobile multimOdal Next generation Applications) [Aneg04] is a representative example of a complete environment for producing several multimodal web applications. It involves a presentation server for a wide range of mobile devices in wireless LAN and mobile phone networks that transforms a single MWUI specification into a graphical or multimodal UI and adapts it dynamically for diverse devices: WAP-phones, Symbian-based smart phones or PocketPC and PDAs. The application design process is based on use cases that allow refining and validating the design of multimodal UI prototypes for each device. These prototypes are further submitted to a heuristic evaluation performed by evaluators with design experience.

2.3.2 Suede

Suede is a speech interface prototyping tool that enables rapid, iterative creation of prompt-response speech 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 speech system and has 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, the speech designer begins to create dialog script examples. After constructing several scrip examples, the designer starts 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. Because the wizard recognizes user responses, no speech recognition or speech synthesis is necessary to test Suede prototypes. During the analysis, designers examine collected test data, deciding how it should influence the next design iteration in order to obtain a more appropriate flow of the interface.

2.3.3 CSLU Toolkit

The CSLU Toolkit was created to provide the basic framework and tools for users to build, investigate and use interactive language systems. These systems incorporate speech recognition, natural language understanding, speech synthesis and facial animation technologies. The applications included in CSLU toolkit are developed using Tcl/TK and the C programming language:

- **RAD** (Rapid application Developer) is an easy to use graphical authoring tool. It allows creating structured dialogues between users and the computer and a wide variety of interactive programs that run both over the telephone and on the desktop. RAD component allows to drag and drop dialogue states onto a canvas, connect 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. It 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 customizes 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.

2.3.4 MOST

MOST (Multimodal Output Specification Platform) [Rous05] is a platform that allows the design of output multimodal systems, involving graphical, vocal and tactile modalities. The design is based on a cycle model composed of three steps: analysis, specification and simulation. After identifying the necessary output interaction components (i.e., mode, modality and medium) in the analysis step, the specification step formalizes the results of the previous step based on a series of attributes and criteria assigned to each specific output interaction component. Further, depending on the current state of the interaction context, a behavioral model allows the identification of the most suitable output form that can be used in order to present each interaction component. The behavioral model is expressed under the form of a set of election rules that produces an adapted multimodal presentation. The simulation step is sustained by a tool that allows the execution of the previously determined output specification.

2.4 Conclusions

This section concludes with a summary of the state of the art. We use this summary in order to establish a list of requirements defined along with the corresponding motivations based on which we select and extend in Section 3 the user interface description language for the development of web user interfaces.

2.4.1 Summary of the state of the art

The current section offers an overview of the languages surveyed in this chapter. Table 1-1 sums up in a comparative analysis the following set of features:

- *Input modalities:* specify the input modalities that can be employed by the end-user in the interaction with the system. For a better understanding of the involved interaction modality we specify:
  - ✓ For graphical modality: the interaction devices (e.g., keyboard, mouse)
  - ✓ For vocal modality: 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 the user with information. We specify:
  - ✓ For graphical modality: the output device (e.g., PC screen, GSM screen)
  - ✓ For vocal modality: the type of vocal 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.
- *Independence of modality:* specifies if in the development life cycle there is a modality-independent level for language specification

- *Separation of modalities:* specifies if the language specifications for the involved modalities are separated or combined
- *Supported CARE properties for input modalities:* specify which are the CARE properties supported by the input modalities
- *Supported CARE properties for output modalities:* specify which are the CARE properties supported by the output modalities
- *Design options:* identifies the existence of design options in the development process of the UIs
- *Model-to-model transformational approach:* indicate the existence of a progressive transformational approach between the models involved in the development process
- *Machine processable vs. Human readable language:* specifies which of these two features have a higher proportion for then language
- *Extensibility for new modalities:* identifies if the language was intended to be extensible with new input and output modalities
- *Development tools:* specifies the name(s) of the development tool(s)
- *Interpretation/Rendering/Converter tools:* identify the interpretation and rendering tools. For some languages converters where developed to target already standardized languages.

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	-
Independence of modality	No	-	Yes	-	No	Yes	-
Separation of modalities	No	-	Yes	-	Yes	Yes	-
Supported CARE properties for input	A, E	-	A, E	-	A, E, R	A, E, R	-
Supported CARE properties for output	A, E, R	-	A, E, R	-	A, E, R	A, E, R	-

Design options	No	No	No	No	No	Yes	No
Model-to-model transformational approach	No	Yes	No	No	No	Yes	No
Machine processable vs. Human readable	Machine processable	Machine processable	Machine processable	Machine processable	Machine processable	Machine processable	Machine processable
Extensibility for new modalities	Yes	No	Yes	No	No	Yes	Yes
Development tool	Galatea Interaction Builder	XIML Validator, Editor, Viewer tools	UIML development tool	IBM WebSphere Voice Toolkit	IBM Multimodal Toolkit	Teresa	No
Interpretation/ Renderer/Conv erter 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	Net Front browser, Opera browser	Teresa (generation of X+V specification)	No

Table 1-1. Comparison of monomodal /multimodal languages

2.4.2 Requirements

When developing user interfaces in general, there are usually 2 approaches taken into account:

A. Rush-to-code approach: is the traditional method adopted by a great number of developers which have the tendency of jumping directly into the generation of the code for their applications without taking into consideration the problems generated by this approach. Some of them are identified below:

- ✓ Completeness: the application will offer to the users a more reduced set of functionalities then expected
- ✓ Consistency: when is ensured it leads to ease of learning, ease of use and improves the user's productivity by leading to higher results and fewer errors because the user can predict what the system will do in any given situation [Niel88].
- ✓ Correctness: the results of the users demands are not the correct ones
- ✓ Errors:
    - Coding: without a good design of the application, coding errors are unavoidable
    - Debugging: the errors are difficult to identify and the recovery from them is hard to achieve a they can be generated either by an incorrect coding or by an inappropriate design of the application due to the rush to code approach
    - Usability: the user will manipulate an application with a reduce ease of use
- ✓ Developing time cost: the sooner is the rush to the code writing, the more time will be spend to develop the application.

The development of Multimodal web user interfaces are submitted to the same type of problems as those summarized above when following the rush-to-code approach. Based on them we define in Table 1-2 a series of criteria according to which we analize the limits of the approach and we localize the impact of each criterion over the designer, programmer or end-user. The estimated time for the development and further modifications (e.g., adding an attribute) considers a reasonably complex task. As it can be observed the end-user is the one that is the most affected by the application of the rush-to-code approach.

Criterion	Rush-to-code approach	Level of impact
Completeness	No guarantee	End-user
Consistency	No guarentee	End-user
Correctness	No guarantee	End-user
Guidance	No	End-user
Coding errors	Yes	Programmer
Debugging errors	Yes	Programmer
Usability errors	Yes	End-user
Estimated time for a first development	Novice user (5 days) Expert user (2 days)	Designer
Estimated time for further modifications	Novice user (1/2 day) Expert user (a coulpe of hours)	Designer

Table 1-2. The Rush-to-code approach

B. Model-based approached: consideres a methodology based on models that are employed in the different development steps and offers guidance for coding complet and correct multimodal web applications. The mehodology (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 Section 2.2. We have grouped them in a decreasing order of importance in: (1) multimodality requirements and (2) ontological and methodological requirements. Each of them is defined along with its corresponding motivation(s):

Multimodality requirements

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

Requirement 2. Support for multimodal output: states that our ontology should allow multiple (i.e., at least two different) output interaction modalities. The requirement is motivated by the definition of the multimodal systems (see Section 1.3.4).

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

Requirement 4. CARE properties support for output modalities: states that our ontology should ensure the support of the CARE properties for output modalities. The current requirement is motivated by the design facilities offered by the CARE properties when defining the relationships that can occur between output modalities.

Requirement 5. Approach based on design space: states that our development life cycle towards a final multimodal web user interface should be supported by a set of design features. This requirement is motivated by the need to clarify the development process in a structured way in terms of options, thus requiring less design effort.

Requirement 6. Openness to new modalities: state that the conceptual structure and the transformations applied on it should allow the extension with new types of interaction modalities. This requirement is motivated by the constant appearance of new computing platforms, each of them coming with a new set of supported interaction modalities. This requirement is a principle that we would like to cover, but we are well aware of the fact that very complex interactions can not be supported.

Requirement 7. Separation of modalities: states that the concepts and the specifications corresponding to each modality should be syntactically separated one from each other. The current requirement is motivated by two aspects: (1) the flexibility in developing applications due to the fact that the specifications for each modality can be developed separately from the other modalities specifications and combined them altogether later, (2) reusability of the specification or part of a specification of a modality in other applications that involve the use of the same modality.

Ontological and methodological requirements

Requirement 8. Ontological independence of modality: states that the provided ontology should ensure a level in the development life cycle that allows specifying a modality-independent UI. This requirement is motivated by the growing number of new interaction devices and, consequently, of interaction modalities that will determine the development of new UIs with new modality capabilities. A modality-independent level will also allow avoiding the redeployment of UIs from scratch. This requirement contributes to the principle of separation of concerns defined in [Dijk76].

Requirement 9. Transformation-based development: states that the development of user interface systems should be based on a successive application of transformations to an initial representation. The current requirement is motivated by the variety of contexts of use (i.e., referred as the triple <user, computing platform, physical environment>) for which a UI is designed [Limb04b]. This variety stresses the need for abstractions from which it is possible to obtain context specific representations by progressive refinements. The advantage of such representations is given by the ability to reason on one single model and obtain many different UIs.

Requirement 10. Machine processable: states that the provided ontology should be proposed in a format that can be legible by a machine. This requirement is motivated by necessity of transposing the ontological concepts into representations that can be processed by machines.

Requirement 11. Human readable: states that the proposed ontology should be legible by human agents. The current requirement is motivated by two aspects: (1) the need of defining 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.

Requirement 12. Ontological 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 13. Reuse of specification: refers to the possibility of reusing hole or part of a specification for another system. The current requirement is motivated by the general principle in software engineering.

Requirement 14. Methodological 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 15. Methodological 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 hinder designers to add, delete, modify and reuse these steps.

Requirement 16. Support for tool interoperability: refers to the possibility of reusing the output provided by one tool into another tool. 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.

CHAPTER 3 CONCEPTUAL MODELLING OF MULTIMODAL WEB USER INTERFACES

After identifying the requirements of multimodal web applications in Chapter 2, the current chapter introduces the concepts of our framework extended for the development of multimodal web applications. Section 3.2 presents the selection of the user interface description language. Sections 3.3, 3.4 and 3.5 describe the semantics, the syntax and the stylistics of the selected language, respectively.

3.1 Selection of the User Interface Description Language

The central goal of the current dissertation is to provide a model-driven approach that can be employed in order to offer designers the capability of developing multimodal web UIs. In software engineering, model-driven approaches rely in the power of models to construct and reason about software systems. The goal of the model-driven approach for user interface development is to propose a set of abstractions, development processes and tools that enable an engineering approach for the development of UIs. In order to support the main goal of the present dissertation a user interface description language is desirable. With respect to this goal we have considered two choices: (1) introducing a new specification language (2) reusing or expanding an already existing user interface description language. Starting from scratch with a specification language requires a lot of efforts before reaching a level of interest that is significant. Thus, the first option appears to be resource-consuming. With respect to the second option we have considered several multimodal languages for which a set of shortcoming have been identified:

- X+V:
  - ✓ Is an implementation language and not a User Interface 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 8. Ontological independence of modality)
  - ✓ There are no design options in the development life cycle (Requirement 3. Approach based on design space)
- XISL:
  - ✓ There is no modality-independent level (Requirement 8. Ontological independence of modality)
  - ✓ The specification language does not specify the interaction modalities separately (Requirement 7. Separation of modalities)
  - ✓ There are no design options in the development life cycle (Requirement 3. 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 (Requirement 9. Transformation-based development), but the transformations are precomputed and hard-coded. Thus, modifiability and extendibility are not supported (Requirement 15. Methodological exendibility).

- ✓ As the transformations are hard-coded, they are not expressed in the same language as the specification language (Requirement 13. Ontological homogeneity).

To the above identified shortcomings we add a general one: if we want to submit an extension of an existing language there is no guarantee that the Consortium which is in charge with that language will consider this extension.

After identifying the shortcomings for the above multimodal languages we also considered UsiXML (USer Interface eXtensible Markup Language) (www.usixml.org), a User Interface Description Language that allows the specification of various types of UIs such as Graphical User Interfaces (GUIs), Vocal User Intefaces (VUIs) and 3D User Interfaces (3D UIs). For the main goal of this dissertation we have selected UsiXML 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 identified in [Calv03]. This framework is a reference for classifying UIs supporting multiple target platforms or multiple contexts of use in the field of context-aware computing and structures the development life cycle into four levels of abstraction: task and concepts, abstract user interface, concrete user interface and final user interface. The identification of the four levels and their hierarchical organization is built on their independence with respect to the context in which the final software system is used. Thus, the Task and Concepts levels is computation independent, the Abstract UI level is modality independent and the Concrete UI level is toolkit independent.

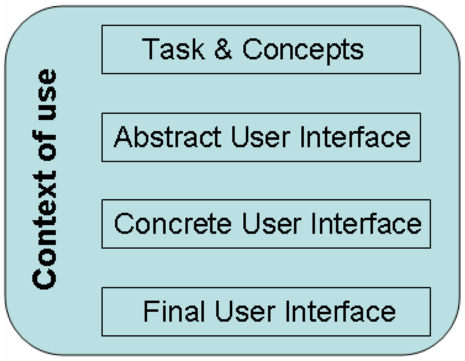


Figure 3-1. The Cameleon reference framework for multi-target UIs

- UsiXML relies on a transformational approach that progressively moves from the Task and Concept level to the Final User Interface (Requirement 9. Transformation-based development).
- The steps of the transformational approach define in a comprehensive way their logic and application [Limb04] (Requirement 14. Methodological explicitness).
- The transformational methodology of UsiXML allows the introduction of new development sub-steps, thus ensuring the possibility to explore alternatives for each sub-step and to add new ones (Requirement 15. Methodological extendibility).
- UsiXML has an underlying unique formalism represented under the form of a graph-based syntax. (Requirement 12. Ontological homogeneity).

- UsiXML allows reusing elements previously described in anterior UIs to compose a UI in new applications. This facility is provided by the underlying XML syntax of UsiXML which allows the exchange of any specification. Moreover, the ability of transforming these specifications with a set of transformation rules increases the possibilities for reusing them (Requirement 13. Reuse of specification).
- 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 and an appropriate formalism for human use (Requirement 11. Human readable).
- UsiXML supports modality independence as UIs can be described at the Abstract UI level in a way that remains independent of any interaction modality such as graphical interaction, vocal interaction or 3D interaction (Requirement 8. Ontological independence of modalities).
- UsiXML supports the incorporation of new interaction modalities thanks to the modularity of the framework where each model is defined independently of the others and to the structured character of the models ensured by the underlying graph formalism (Requirement 6. Openness to new modalities).
- UsiXML is supported by a collection of tools that allow processing its format (Requirement 10. Machine processable).
- UsiXML allows cross-toolkit development of interactive application thanks to its common UI description format (Requirement 16. Support for toolkit interoperability).

In the following sections we present the UsiXML models and their corresponding underlying concepts, for which UML class diagrams are used. The description is based on [USIX06] and emphasizes the improvements and expansions realized in order to adapt the UsiXML models to the requirements of multimodal web UIs.

3.1.1 Task Model

The *Task Model* describes the interactive tasks as viewed by the end user interacting with the system. The task model is expressed here according to our extended version of ConcurTaskTree notation [Pate97]. The Task Model (Figure 3-2) is composed of *tasks* and *task relationships*. *Tasks* are, notably, described with attributes like *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* refers to four basic types of tasks: user's, interactive, system and abstract task. For leaf task we consider two attributes (i.e., *userAction* and *taskItem*) that enable a refined expression of the nature of the task. This expression is based on the taxonomy introduced by [Cons03] that allows to qualify a UI in terms of abstract actions it supports. The *userAction* is represented by a verb that indicates a user action required to perform the task and the *taskItem* which refers to a type object or subject of an action. The possible values and their associated definition are presented in Section 3.1.2.

Task relationships are relationships involving several occurrences of different (or the same in some cases) tasks. Task relationships are of two main types:

- *Decomposition:* enables to represent a hierarchical structure of the task tree. Decomposition relationship is implicit within the XML syntax of the language and it is represented by simple embedding of elements.

- *Temporal:* allows specifying temporal relationships between tasks. We use LOTOS operators as they have been applied to task modeling in [Pate97].

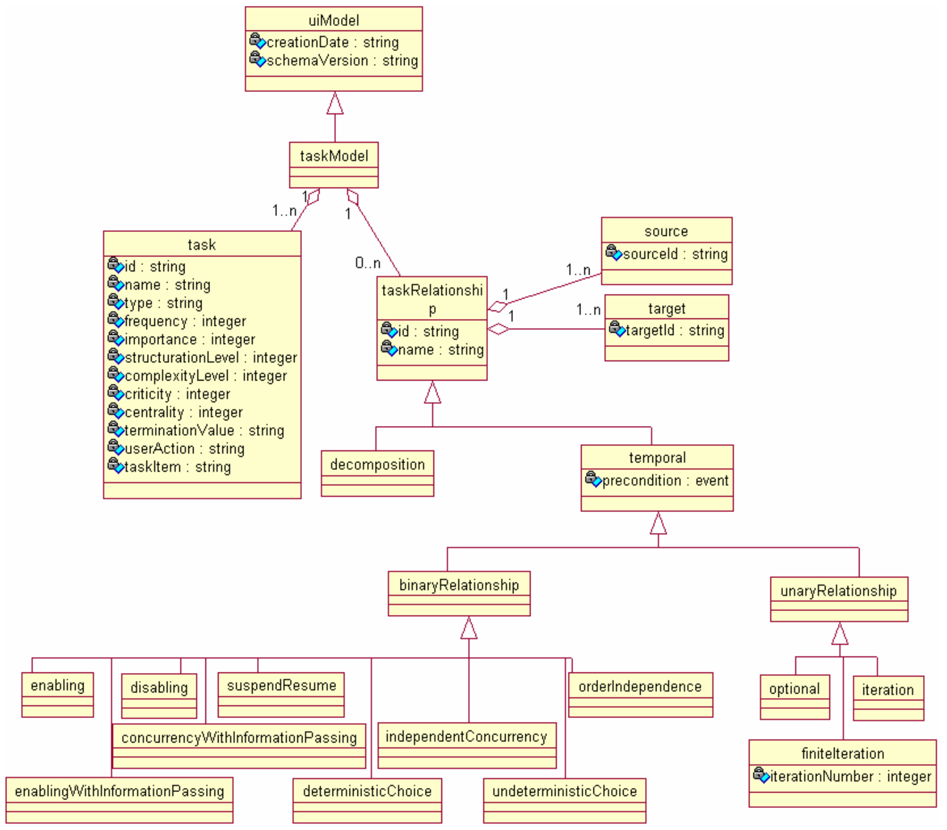


Figure 3-2. Meta-model of the Task Model

3.1.2 Domain Model

The Domain Model (Figure 3-3) is a description of the classes of objects manipulated by a user while interacting with a system. It consists of one or many *domainClasses*, and potentially one or many *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 level of a class are the following: *attributes*, *methods*, and *objects*. An *attribute* is a particular characteristic of a class. Attributes are further described by the elements constituting the attribute class. The *attributeDataType* refers to basic data types as string, integer, real, boolean or enumerated. An *enumeratedValue* describes in extension an attribute that has the characteristic of being enumerated. The *attributeCardMin* and *attributeCardMax* describes, respectively, the lower and upper bound of the attribute cardinality (0 means that the attribute is not mandatory, 1 means that it is mandatory). A *method* is the description of a process able to change the system's state. Here, the methods are described by its signature (i.e., its name, input and output parameter(s)). An *object* is an instance of a class and is composed of attribute instances and can call methods.

A *domainRelationship* describes various types of relationships between classes. They can be classified in three types: generalization, aggregation, ad hoc. Class relationships are described with several attributes that enable to specify its role names and cardinalities.

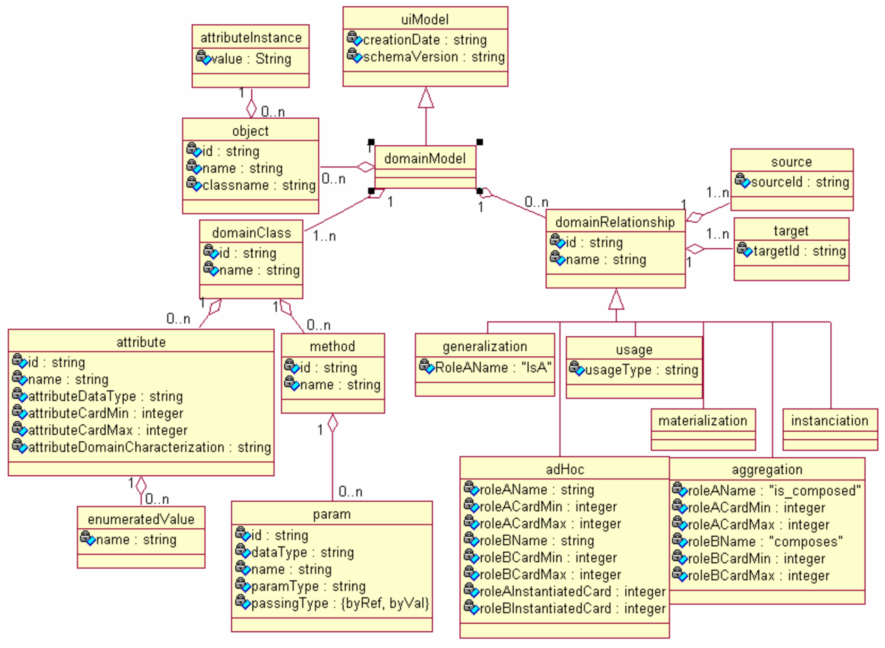


Figure 3-3. Meta-model of the Domain Model

3.1.3 Abstract User Interface Model

Abstract User Interface (AUI) Model is a model that 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. As an AUI does not refer to any particular modality, we do not know yet how this abstract description will be concretized: graphical, vocal or multimodal. This is achieved in the next level.

AUI Model (Figure 3-4) is populated by *Abstract Interaction Objects (AIO)* and A*bstract User Interface Relationships* between them.

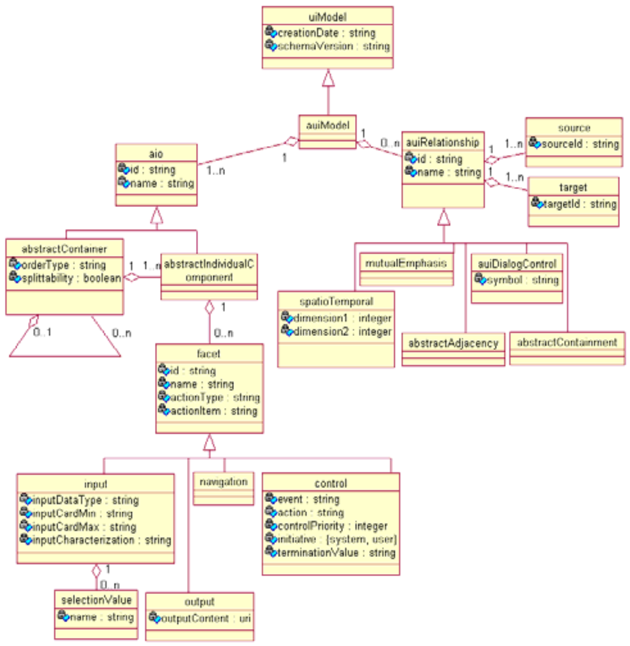


Figure 3-4. Meta-model of the AUI Model

An *AIO* is an element populating an AUI model consisting in an abstraction of widgets found in graphical toolkits (e.g., windows, buttons) and vocal toolkits (e.g., prompts, vocal menus). It can be of two types: *Abstract Individual Component (AIC)* or *Abstract Container (AC).*

An *AIC* is any individual element populating an AC. An AIC assumes at least one basic system interaction function described as *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:* describes the type of input that may be accepted by an AIC
2. *Output facet:* describes what data may be presented to the user by an AIC
3. *Navigation facet:* describes the possible container transition a particular AIC may enable
4. *Control facet:* describes possible methods from the Domain Model that may be triggered from a particular widget.

An AIC may assume several facets simultaneously. For instance an AIC may display an output while accepting an input from a user, 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 possible values (Table 3-1) are the same as for the *userAction* attribute of a task from the Task Model. The *view* value allows in [USIX05] to express an information by displaying it and can be reified in the Concrete level only by a graphical object. In order to keep the AUI Model independent of any modality, we replace this value by introducing in [USIX06] *convey*, a more appropriate value for actionType attribute, as it does not specify the employed modality.

actionType	Definition
start/go	Specifies that the AIC triggers an action
stop/exit	Specifies that the AIC puts an end to an action
select	Specifies that the AIC allows a selection action over multiple options
create	Specifies that the AIC is creating an item
delete	States that the AIC is dedicated to the deletion of items
modify	States that the AIC is dedicated to the modification of items
move	States that the AIC allows the movement of an item
duplicate	States that the AIC allows the creation of copies of an item
toggle	States that the AIC specifies the existence of two different states of an item
convey	**States that the AIC expresses an information without specifying the employed modality: graphical, vocal, etc.[USIX06]**

Table 3-1. Definition of possible values for the actionType attribute of a facet

The *actionItem* characterizes the item that is manipulated by the AIC. The possible values (Table 3-2) are identical to those of *taskItem* attribute of a task from the Task Model.

actionItem	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 of	Specifies that the item is composed of a list of elements

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

Table 3-2. Definition of possible values for the actionItem attribute of a facet

By combining these two attributes a series of possible cases will appear. Table 3-3 exemplifies several possible associations.

actionType	actionItem	Example
start	operation	Search a definition of a word in an online dictionary
stop	operation	Stop searching the definition
select	element	Select the gender of a person
create	element	Input a new email address in a form
delete	collection of elements	Erase a list of phone numbers
modify	collection of containers	Modify a list of addresses
move	element	Drag and drop a predefined shape from a toolbar to the working area
duplicate	collection of elements	Copy the coordinates of a person (name, email, fax, phone number)
toggle	element	Switching on/off the connection with a network
convey	**element**	**Express the result of a computational operation (the result can be expressed graphically by displaying it on the screen or vocally by system utterance)**
convey	**container**	**Express the starting date of a conference (the day, month and year can be displayed on the screen or can be uttered by the system)**
convey	**collection of elements**	**Express the authors list of a book (the list of authors can be displayed or can be uttered by the system)**
convey	**collection of containers**	**Express the staring and ending date of a conference (the day, month and year of the staring date and, respectively the ending date can be displayed or can be uttered by the system)**

Table 3-3. Examples of combinations between actionType and actionItem attribute values

AUI Relationships are abstract relationships among AUI objects. Relationships may have multiple sources and multiple targets. There are a couple of types of relationships, between 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 a dialog control in terms of LOTOS operators between AIOs.

3.1.4 Concrete User Interface Model

Concrete User Interface (CUI) Model is a model that allows the specification of the presentation and behavior of a UI with elements that can be perceived by the users [Limb04b]. The CUI abstracts a Final UI in a definition that is independent of programming toolkit peculiarities.

CUI Model (Figure 3-5) 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 widgets 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 that users can perceive and/or manipulate (e.g., window, push button, text field, check box, vocal output, vocal input, vocal menu). Because UsiXML considers both graphical and vocal modalities, CIOs are further divided into two types: *graphicalCIOs* and *vocalCIOs*. A detailed explanation regarding the types of *graphicalCIOs*, *vocalCIOs* and the corresponding *Concrete User Interface Relationships* between them, along with their semantics and syntax is presented in the following sections.

Any CIO can have any number of *behaviors*. A *behavior* is the description of the triplet *event-action-condition* that determines the UI change. In the following we offer a brief description for each of the terms involved in the triple, but a more detailed documentation can be found in [USIX06]:

- *Event:* specifies an expression triggering one or several actions. Events are restricted to a specific event language. Graphical *eventsTypes* are described in Section 3.2.1, while the vocal ones are identified and defined in Section 3.2.2. The attribute *eventContext* allows mentioning the concerned CIO, depending on the type of event. The attribute *device* is a reference to the device with which the event is triggered.
- *Action:* is a process triggered by an event performed on a CIO. An action may be a method call, a UI internal change, etc.
- *Condition:* enables to specify a pre/post-condition attached to an action. A condition is expressed as a graph patters (i.e., a rule term) that must be fulfilled in the specification before or after the application of an action. Conditions may be combined with Boolean operators to compose complex conditions.

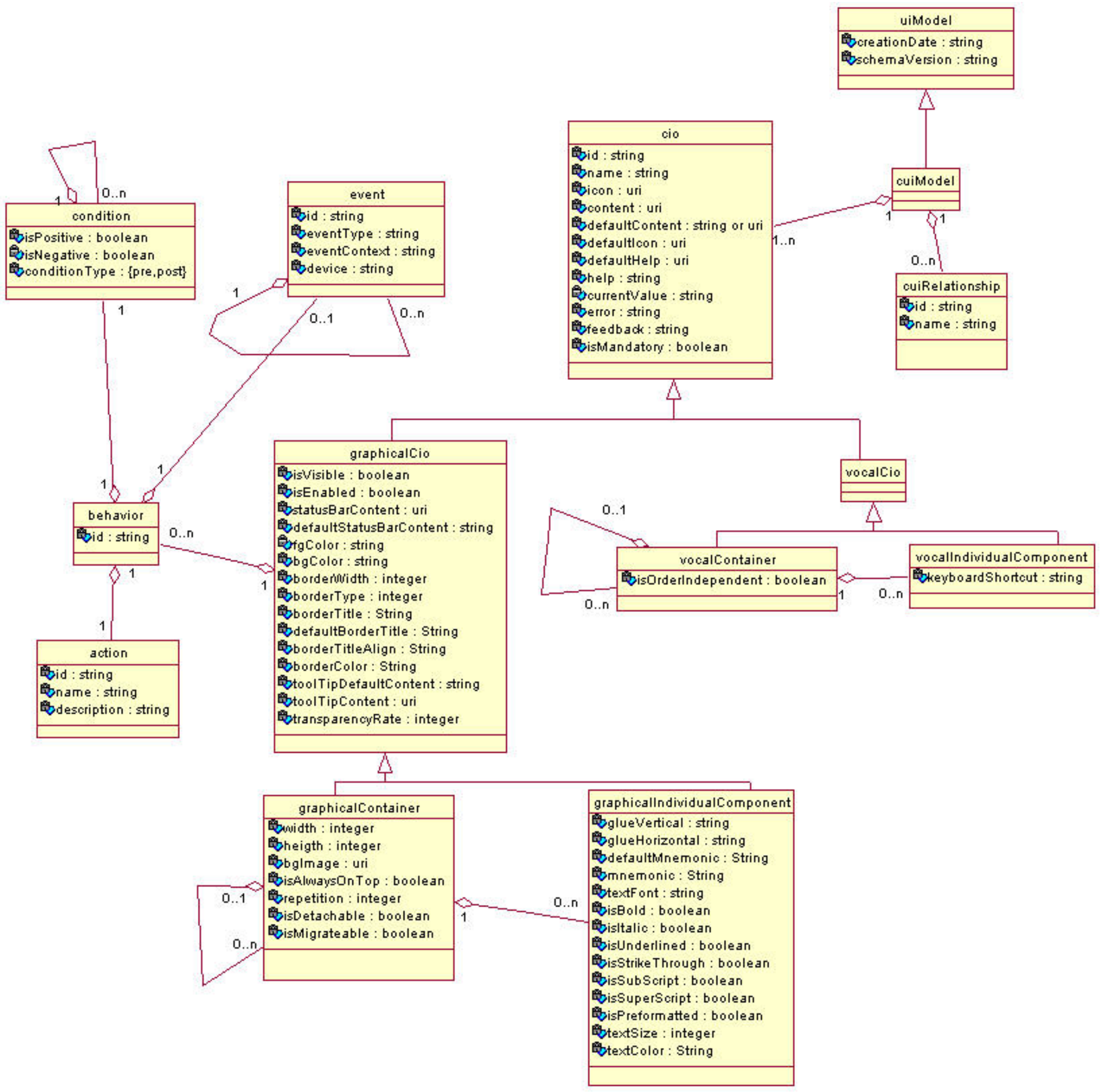


Figure 3-5. Excerpt of the CUI Meta-model

3.1.5 Final User Interface

The *Final UI (FUI)* is the operational UI, i.e. any UI running on a particular computing platform either by interpretation (e.g., through a web browser) or by execution (e.g., after the compilation of code in an interactive development environment).

3.1.6 Context Model

The *Context Model* (Figure 3-6) describes all the entities that may influence how the user's task is carrying out with the future UI. It takes into account three relevant aspects, each aspect having its own associated attributes: *user type* (e.g., experience with device and/or system, task motivation), *computing platform type* (e.g., desktop, PocketPC, PDA, GSM), and *physical environment type* (e.g., lighting level, stress level, noise level). These attributes initiate transformations that are applicable depending on the current context of use.

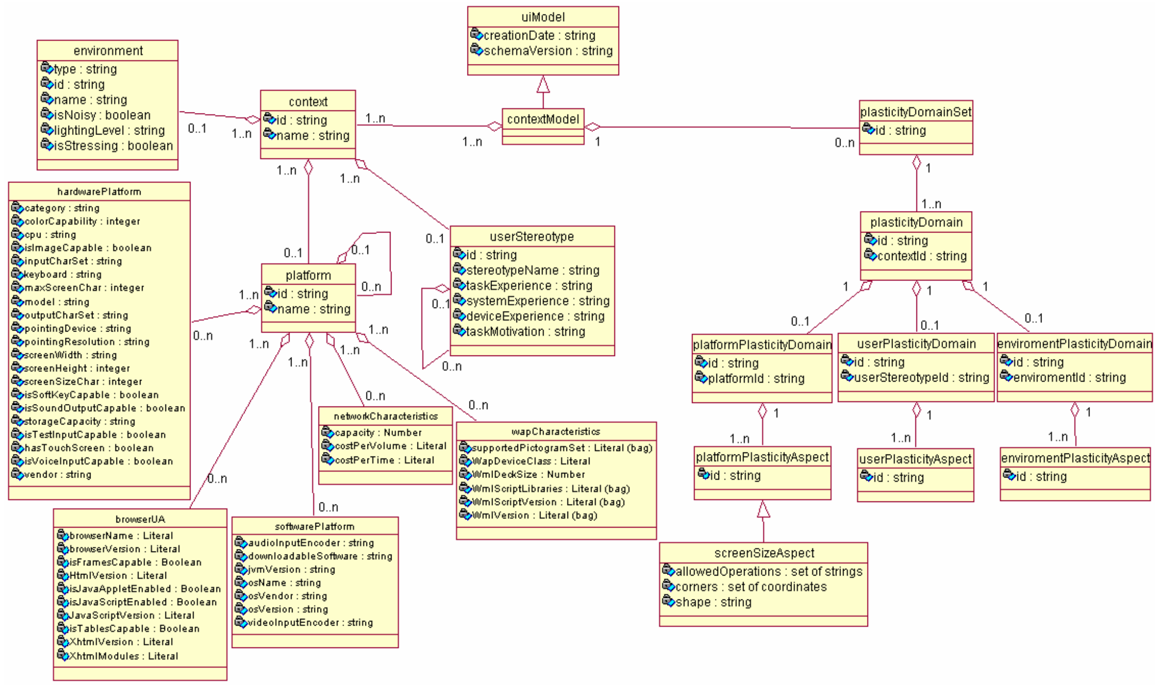


Figure 3-6. Meta-model of the Context Model

3.1.7. Mapping Model

The *Mapping Model* (Figure 3-7) contains a series of related mappings between models or elements of models. A mapping model serves to gather a set of pre-defined, inter-model relationships that are semantically related. It consists of one to many *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 UI component (at abstract or concrete level) and a domain attribute or instantiated attribute (at run time). Updates 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 UI individual component (either at the abstract or at the concrete level)
- *IsExecutedIn:* indicates that a task is performed through one or several ACs and AICs.
- *IsReifiedBy:* maps the elements of an AUI onto elements of a CUI. This relationship specifies how any AIO can be reified by a CIO.

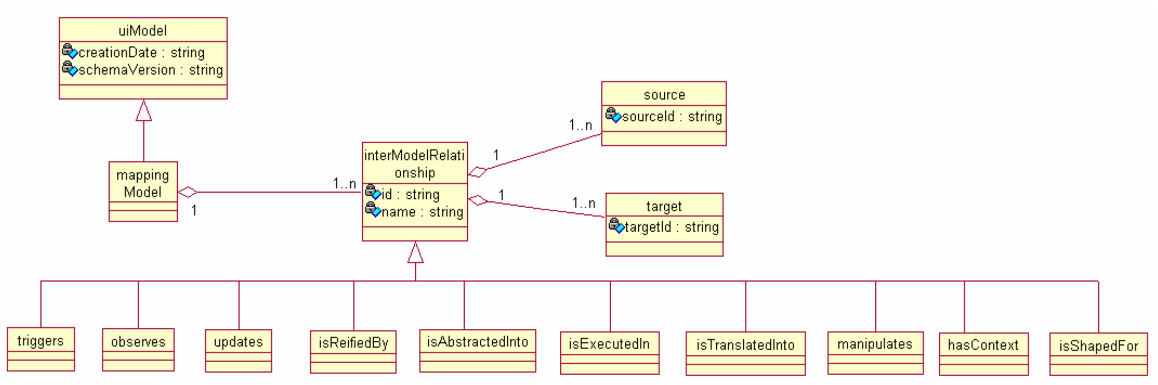


Figure 3-7. Meta-model of the Mapping Model

3.1.8 Transformation Model

Transformation Model (Figure 3-8) contains a set of rules enabling the transformation of one specification (at a certain level of abstraction) into another or to adapt a specification for a new context of use. A transformation rule realizes a unit transformation operation on a model. It is composed of a:

- *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 have to hold false before trying to match LHS into the host model
- *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 realized by one (and only one) 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, for instance choosing widgets, defining navigation structure, etc

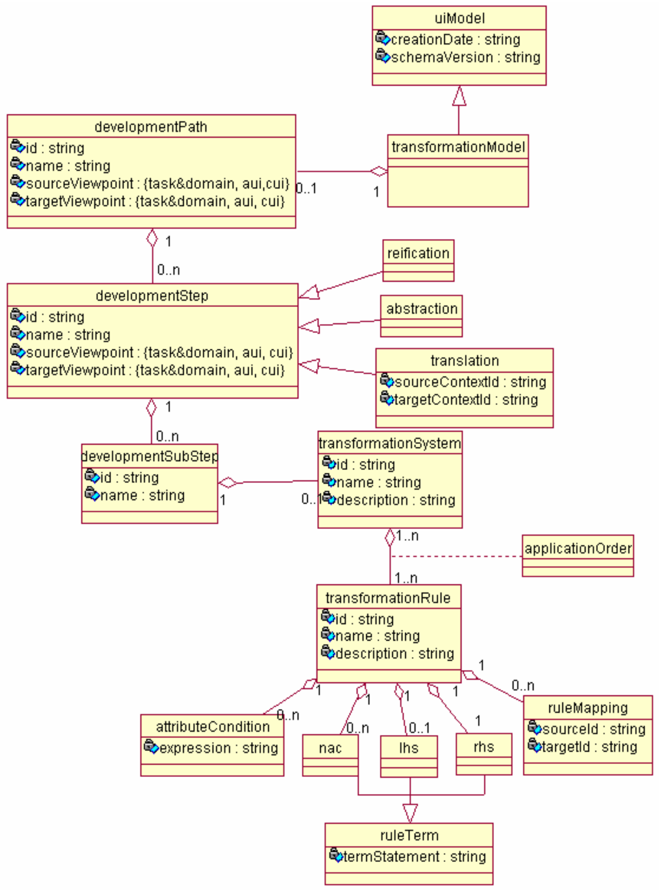


Figure 3-8. Meta-model of the Transformation Model

3.2 Semantics

Semantics (in Latin letters semantikós, or *significant meaning*, derived from sema, *sign*) is the study of meaning, in some sense of a term.

In this subsection we provide the semantics for the *graphicalCIOs, vocalCIOs* and *multimodalCIOs* and the corresponding relationships between the involved objects which are part of our transformational approach for the development of multimodal web UIs. The semantics of the objects is defined based on [USIX06].

3.2.1 Graphical Concrete Interaction Object semantics

Graphical CIOs are divided into Containers and Individual Components.

*Graphical Containers (GCs) (*Figure 3-9) contain a collection of CIOs (either GICs or GCs) that support the execution of a set of logically/semantically connected tasks. We define here the semantics of several of the most used types of containers:

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

- *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 graphicalContainer.
- *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 that appears when an item is selected. The selected item is generally at the top of the display screen, and the menu appears just below it.

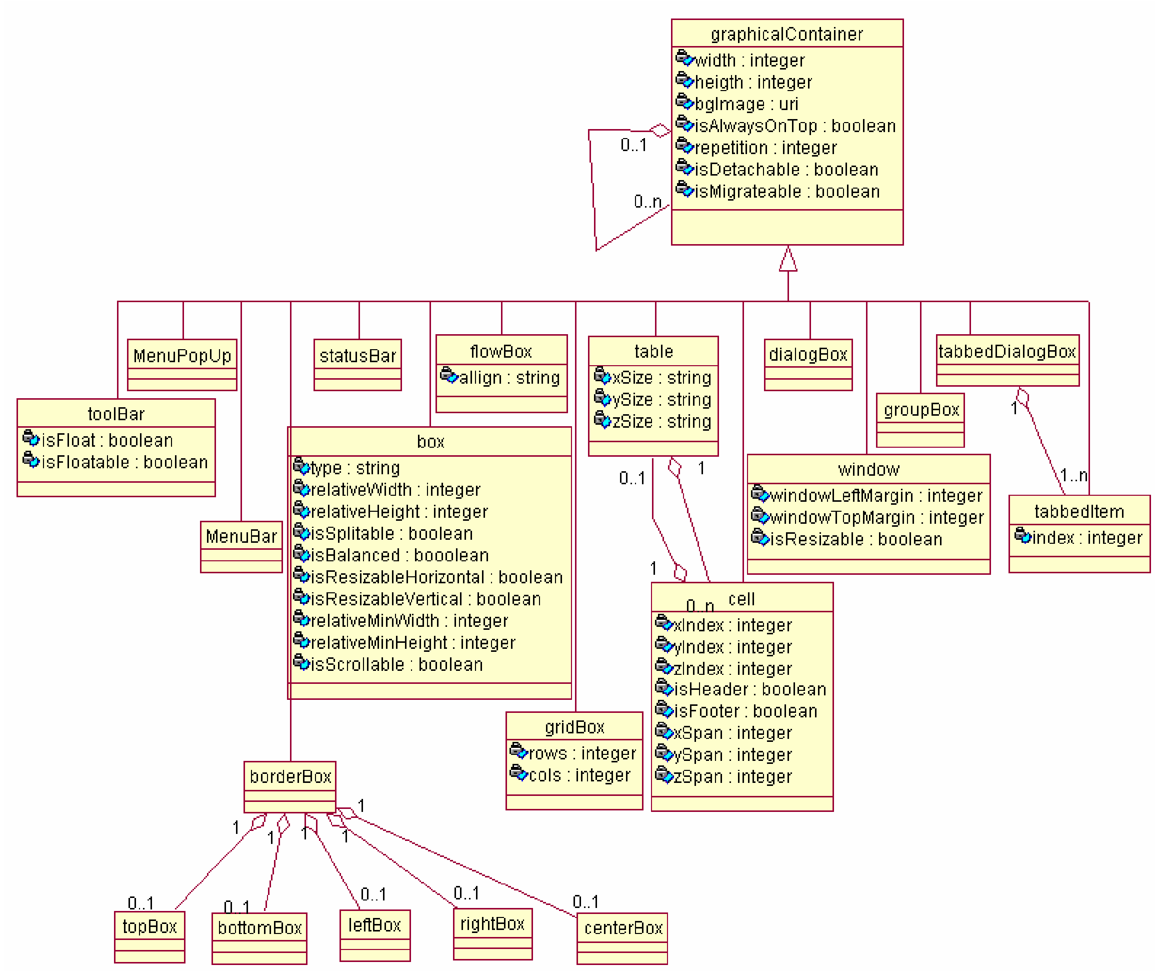


Figure 3-9. Graphical containers

Graphical Individual Components (GICs) are objects contained in GCs. They realize an abstraction of widgets found in most popular graphical toolkits (e.g., Java AWT/Swing, HTML 4.0, Flash DRK 6). Figure 3-10 shows a part of GICs defined in [USIX06] which is an improved version of [USIX05]. This new version offers more graphical widgets and attributes and a more clear difference between them. For instance, in order to distinguish between components that allow handling input and output textual content, the *textComponent* object from [USIX05] has been splitted into *inputText* and *outputText*. For a full definition of all the GICs and their associated attributes please refer to [USIX06]. In the following we offer the semantics of several types of GICs:

- *InputText:* is a GIC specialized for handling input textual content
- *OutputText:* is a GIC specialized for handling output textual content
- *Button:* is alternatively called trigger button as its aim is 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. An optionButton may be differentiated from a checkBox by the fact that when grouped optionButton selection is mutually exclusive while checkBox allows multiple choices.
- *ComboBox:* enables a direct selection over a collection of sequentially, predefined items. It might also enable editing new items.
- *ImageComponent:* is a GIC specialized for handling image content.

The classification of a set of typical events specified by the attribute *eventType* of the event class (Figure 3-5) is done in [Limb04b]. Two categories of event types are identified: *system events* and *graphical user interface events.*

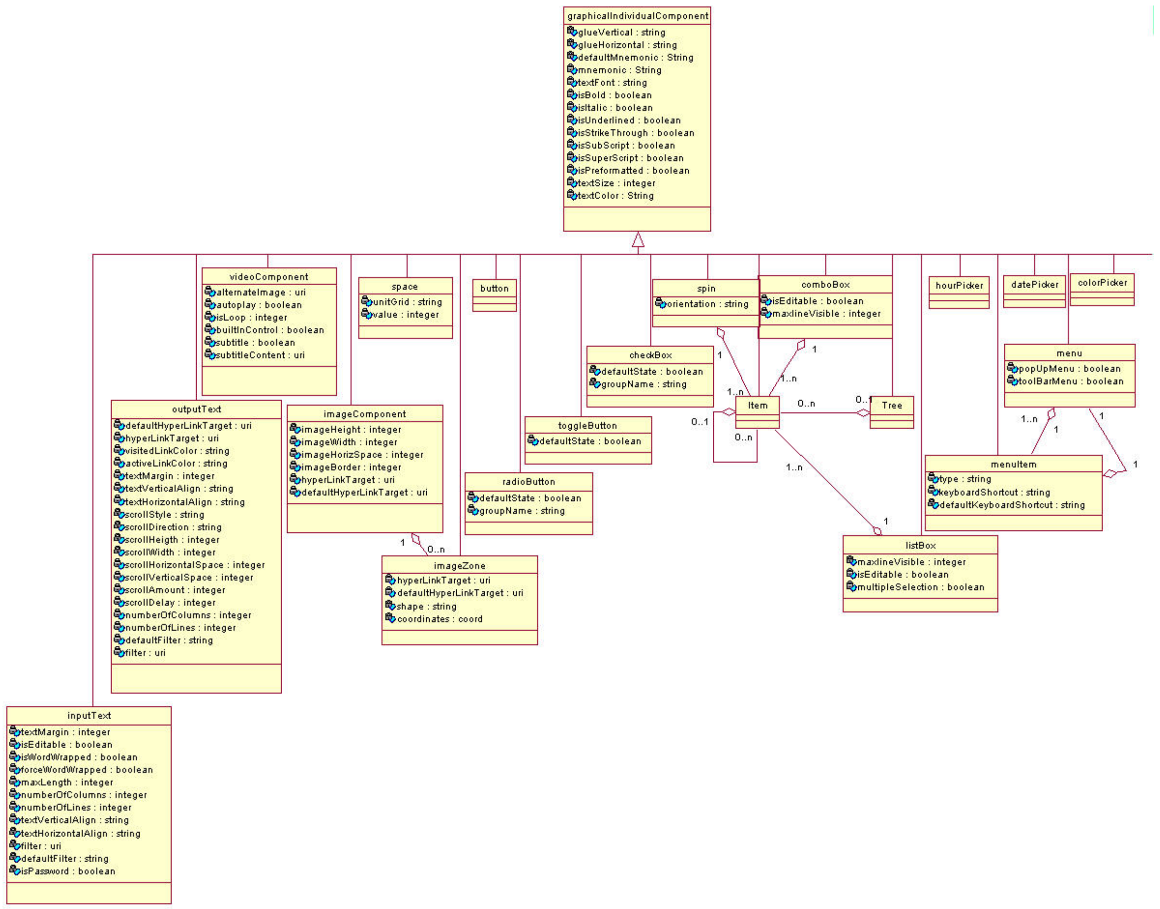


Figure 3-10. Some Graphical Individual Components

3.2.2 Vocal Concrete Interaction Object semantics

[USIX05] supports the vocal modality through the definition of *Auditory Interaction Objects* which can be *Auditory Containers* or *Auditory Individual Components*. *Auditory Containers* represent a logical grouping of other containers or *Auditory Individual Components*. *Auditory Individual Components* are of two types: *auditoryOutput* which may consist in 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 her voice or any other physical device able to produce sound. Auditory relationships are also provided and are of two types:

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

The definitions offered in [USIX05] concerning the vocal elements suffer from a series of shortcomings:

- It does not provide a specialized container that enables a dialog between the system and the end-user (i.e., synthesize data from the system and/or collects data from the user). This is helpful in order to better differentiate from vocal containers that act just as basic containers for all containers and individual components
- It does not provide a specialized vocal container that allows users to choose between different options. This is extremely useful as a usual vocal dialog consists often in multiple choice questions
- It does specify explicitly the type of the system's output. The output could provide the user with prompt information that is synthesized or could provide with some feedback following a previously processed input
- It does not allow to interrupt the execution of the current container or even of the entire application. For instance the end-user would like to put end to a sub-dialog which does not provide her with any useful information or to stop interacting with the application due to an unexpected outer system task.

Based on the requirements specified above, in [USIX06] we have redeveloped the vocal part described in [USIX05] by offering a more detailed set of vocalCIOs (Figure 3-11) that covers the requirements of vocal and multimodal UIs (Requirement 1. Support for multimodal input and Requirement 2. Support for multimodal output)). As graphicalCIOs, the vocalCIOs are divided into Containers and Individual Components. *Vocal Containers (VCs)* represents a logical grouping of other VCs or VICs. There are several types of VCs. All of them inherit the *isOrderIndependent* attribute which indicates if the inputs of the container can be filled in any order or not:

- *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 data from the system and/or collect data from the user.
- *VocalMenu:* is a VC that allows to choose among different vocalMenuItems.
- *VocalConfirmation:* is a VC that requests from the user a confirmation of a previous input. 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. The possible values are {0-9, #, *}. VICs are of six types:

- *VocalOutput:* is an object used to synthesize data to the user. This data is specified in the attribute *defaultContent* inherited from the CIO class. An attribute *volume* 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 sudevided into:
  - ✓ *VocalFeedback:* is a vocalOutput providing the user with some feedback following a previously processed vocalInput. For example: "Your answer was: male".
  - ✓ *VocalPrompt:* is a vocalOutput providing the user with prompt information that will be synthesized. If there is an audio file to be played, the attribute audioSource specifies its URI.
  - ✓ *VocalMenuItem:* is a vocalOutput presenting 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 *nextForm* attribute specifies the reference to the vocalForm attached to this vocalMenuItem either as a string (the id of the vocalForm) or as a URI (an external reference).
- *VocalInput:* is an object used to gather input from the user by speech recognition. The *elapsedTime* attribute is the input duration in which the user is allowed to utter the input. The duration is expressed in seconds. An attribute *grammar* specifies a set of utterances that a user may speak in order to perform an action or to supply an information.
- *VocalNavigation:* specifies a transition to another vocalForm. The *navigationType* attribute defines the navigation type. Allowed values are link, submit and goto. An attribute *isBridgeable* indicates if the source document remains active during the navigation.
- *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* Class (Figure 3-5):

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

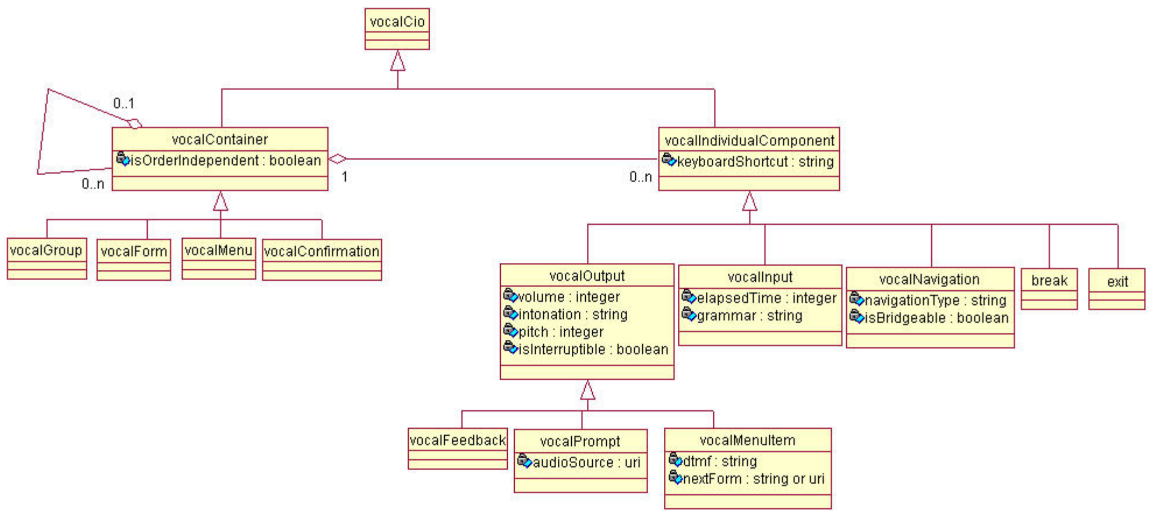


Figure 3-11. Vocal Interaction Objects

For a better understanding of the concepts defined above we exemplify graphically two possible vocal interactions between the system (S) and the end-user (U). The first dialog (Figure 3-12) describes the fulfillment of the **Provide age** task by an end-user. The involved vocalCIOs are described in the order of dialog flow:

- **VocalForm:** is the VC that contains all the vocalCIOs involved in the dialog
- **VocalPrompt:** is the VIC used to invite the user to input her age
- **VocalInput:** is the VIC that gathers the user's input (the age) by speech recognition
- **VocalConfirmation:** is the VC which solicits the confirmation of the introduced age
- **VocalFeedback:** is the VIC that provides the user with the feedback regarding the previously introduced age
- **VocalPrompt:** is the VIC used to invite the user to confirm the previously provided feedback
- **VocalInput:** is the VIC that gathers the user confirmation by speech recognition

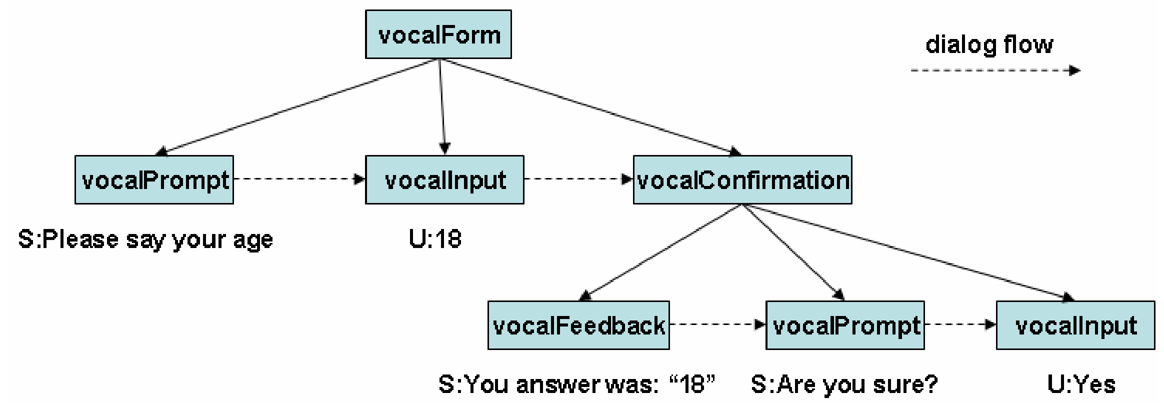


Figure 3-12. VocalCIOs involved in the fulfillment of **Provide age** task

The second dialog (Figure 3-13) describes a virtual shop where users can buy ice cream. It consists of two basic sub-tasks: first the user provides her name to the system and the she chooses her favorite flavor. 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 that contains the vocalCIOs involved in the fulfillment of first sub-task
- **VocalPrompt:** is the VIC used to welcome the user to the virtual ice cream shop and to invite her to input the name
- **VocalInput:** is the VIC that gathers the user's input (the name) by speech recognition
- **VocalMenu:** is the VC that allows to choose between different flavors of the icecream
- **VocalMenuItem1:** is the VIC used to specify the first flavor (vanilla)
- **VocalMenuItem2:** is the VIC used to specify the second flavor (chocolate)
- **VocalMenuItem3:** is the VIC used to specify the third flavor (lemon)
- **VocalInput:** is the VIC that gathers the user's input (the favorite flavor) by speech recognition
- **VocalFeedback:** is the VIC that provides the user with the feedback regarding the previously introduced flavor

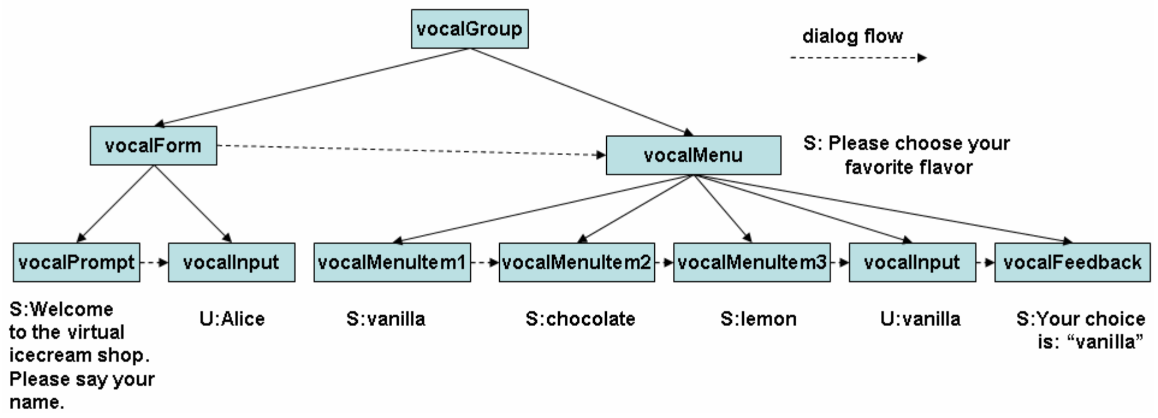


Figure 3-13. VocalCIOs used for a Virtual ice cream shop

3.2.3 Multimodal Concrete Interaction Object semantics

MultimodalCIOs are obtained as a result of the combination between *graphicalCIOs* and *vocalCIOs*.

Table 3-4 identifies for a set of popular widgets a possible correspondence with CIOs. This identification is done for three types of UIs (i.e., graphical, vocal, multimodal). A correspondent rendering for each of them is illustrated also. For multimodal user interfaces we consider the guidance (ensured by representative icons), a design option detailed in Section 4.3.3.2.

1. **Label:**
   - **Graphical interaction:** ensured by outputText.
   - **Vocal interaction:** ensured by vocalPrompt.
     Example: the system is uttering: "Welcome to the UCL web site".
   - **Multimodal interaction:** ensured by outputText and vocalPrompt

Example: the outputText contains the underlined word "Welcome". By clicking on it the system will utter (vocalPrompt) the entire welcome text: "Welcome to the UCL site".

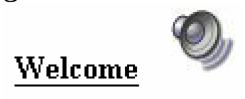
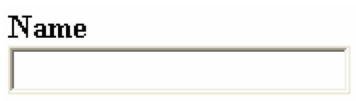
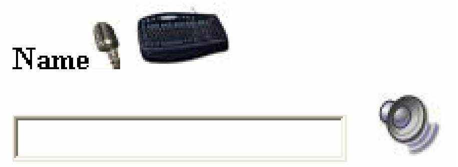
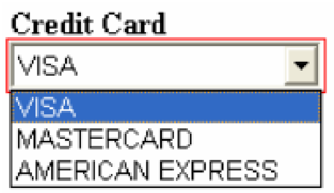
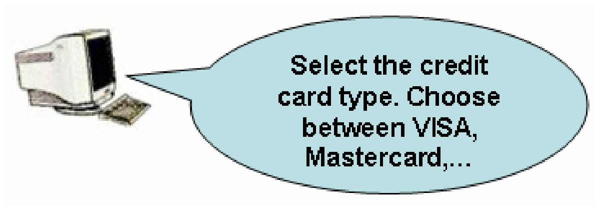
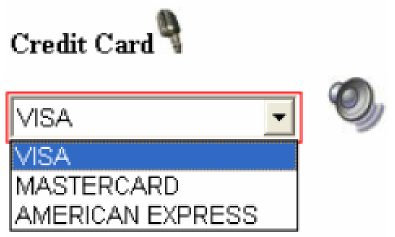
2. **Label + Text field:**
    - **Graphical interaction:** ensured by outputText and inputText.
    - **Vocal interaction:** ensured by vocalPrompt and vocalInput.
      Example:
        - ✓ Computer (vocalPrompt): "Please say your name"
        - ✓ User (vocalInput): "Andy Garcia"
    - **Multimodal interaction:** ensured by outputText, inputText, vocalInput and guidance (representative icons).
      Example:
        - ✓ The user is inputting his name using the microphone (vocalInput): "Please say your name"
        - ✓ The system recognizes the input and displays it (inputText): "Andy Garcia"
        - ✓ The system is providing also a feedback (vocalFeedback) of the recognized input: "Your name is Andy Garcia".

3. **Label + Combo box:**
    - **Graphical interaction:** ensured by an outputText and a comboBox with items.
    - **Vocal interaction:** ensured by vocalMenu with vocalItems and vocalInput.
      Example:
        - ✓ Computer (vocalMenu): "Select the credit card type. Choose between (vocalItems) Visa, MasterCard **or** American Express."
        - ✓ User (vocalInput): "Visa"
        - ✓ Computer (vocalFeedback): "Your choice is: Visa."
    - **Multimodal interaction:** is ensured by the following CIOs: outputText, comboBox with items, vocalMenu with vocalItems, vocalInput and guidance (representative icons).
      Example:
        - ✓ The user clicks on the Credit Card label (outputText)
        - ✓ The system invites the user to choose between different credit cards by vocally outputting the available possibilities (vocalMenu with vocalMenuItems): "Select the credit card type. Choose between (vocalItems) Visa, MasterCard **or** American Express."
        - ✓ The user in inputting his type of credit card using the microphone (vocalInput): "Visa"
        - ✓ The system graphically displays the recognized input (comboBox)
        - ✓ The system also provides a feedback (vocalFeedback) of the recognized input: "Your choice is: Visa."

4. **Group of radio buttons:**
    - **Graphical interaction:** ensured by a groupBox embedding a set of radioButtons.
    - **Vocal interaction:** ensured by a vocalMenu with vocalItems and a vocalInput.
      Example:
        - ✓ Computer (vocalMenu): "Please say your gender. Choose between (vocalItems) male and female."
        - ✓ User (vocalInput): "Male"
    - **Multimodal interaction:** is ensured by the following CIOs: a groupBox embedding a set of radioButtons, a vocalMenu with vocalMenuItems, a vocalInput and guidance (representative icons).
      Example:
        - ✓ The user clicks on the Gender label (the name of the groupBox)
        - ✓ The system invites the user to choose his gender by vocally outputting the available possibilities (vocalMenu with vocalMenuItems): "Please say your gender. Choose between male and female."
        - ✓ The user is uttering his gender using the microphone (vocalInput): "Male"
        - ✓ The system graphically displays the recognized input by checking the corresponding radioButton.

5. **Group of check boxes:**
    - **Graphical interaction:** ensured by a groupBox embedding a set of checkBoxes.
    - **Vocal interaction:** ensured by a vocalMenu with vocalItems and a vocalInput.
      Example:
        - ✓ Computer (vocalMenu): "Please select your hobbies. Choose one or more of the following: sport, travel, music, movies."
        - ✓ User (vocalInput): "Sport and music"
    - **Multimodal interaction:** is ensured by the following CIOs: a groupBox embedding a set of checkBoxes, a vocalMenu with vocalMenuItems, a vocalInput and guidance (representative icons).
      Example:
        - ✓ The user clicks on the *Hobbies* label (the name of the groupBox)
        - ✓ The system invites the user to choose his hobbies by vocally outputting the available possibilities (vocalMenu with vocalMenuItems): "Please select your hobbies. Choose one or more of the following: sport, travel, music, movies."
        - ✓ The user is uttering his hobbies using the microphone (vocalInput): "Sport and music"
        - ✓ The system graphically displays the recognized input by checking the corresponding checkboxes.

6. **Label + List box:**
    - **Graphical interaction:** ensured by an outputText and a listBox with items.
    - **Vocal interaction:** ensured by a vocalMenu with vocalItems and a vocalInput.
      Example:
        - ✓ Computer (vocalMenu): "Please choose your favorite singers: Chris Hay, Lee Hardy, Paul Sheerin,..."
        - ✓ User (vocalInput): "Lee Hardy"
    - **Multimodal interaction:** is ensured by the following CIOs: an outputText, a listBox with items, a vocalMenu with vocalItems, a vocalInput and guidance (representative icons).
      Example:
        - ✓ The user clicks on the *Singers* label (outputText)
        - ✓ The system invites the user to choose his favorite singers by vocally outputting the available possibilities (vocalMenu with vocalMenuItems): "Please choose your favorite singers: Chris Hay, Lee Hardy, Paul Sheerin,..."
        - ✓ The user is uttering his favorite singer(s) using the microphone (vocalInput): "Lee Hardy"
        - ✓ The system graphically displays the recognized input by checking the corresponding item.

Widgets	**User Interface type**		
	Graphical interaction	**Vocal interaction**	**Multimodal (graphical and vocal) interaction**
1. Label	outputText **Welcome to the UCL web site**	vocalPrompt Welcome to the UCL web site	outputText + vocalPrompt + guidance 
1. Label + Text field	outputText + inputText 	vocalPrompt + vocalInput Please say your name	outputText + inputText + vocalInput + vocalFeedback + guidance 
3. Label + Combo Box	outputText + comboBox + items  Credit Card VISA VISA MASTERCARD AMERICAN EXPRESS	vocalMenu + vocalItems + vocalInput  Select the credit card type. Choose between VISA, Mastercard,...	outputText + comboBox + items + vocalMenu + vocalItems + vocalInput + vocalFeedback + guidance  Credit Card VISA VISA MASTERCARD AMERICAN EXPRESS
4. Group of radio buttons	groupBox + radioButtons	vocalMenu + vocalItems + vocalInput	groupBox + radioButtons + vocalMenu + vocalItems + vocalInput + guidance

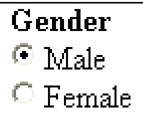
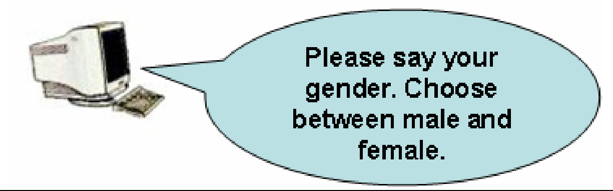
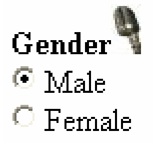
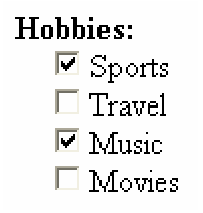
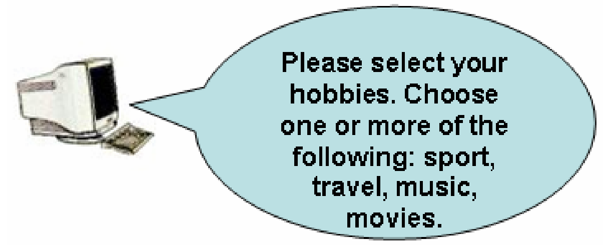
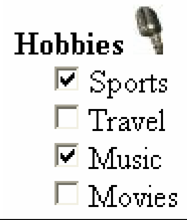
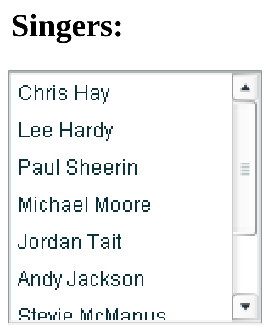
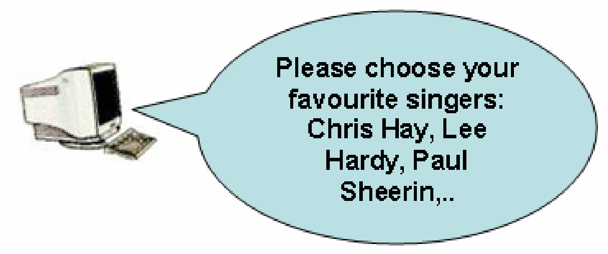
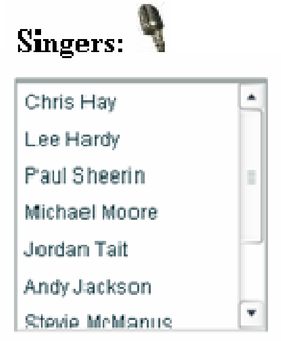
			
5. Group of check boxes	groupBox + checkBoxes 	vocalMenu + vocalItems + vocalInput  Please select your hobbies. Choose one or more of the following: sport, travel, music, movies.	groupBox + checkBoxes + vocalMenu + vocalItems + vocalInput + guidance 
6. Label + List Box	outputText + listBox + items  Singers: Chris Hay Lee Hardy Paul Sheerin Michael Moore Jordan Tait Andy Jackson Stevie McManus	vocalMenu + vocalItems + vocalInput  Please choose your favourite singers: Chris Hay, Lee Hardy, Paul Sheerin,..	outputText + listBox + items + vocalMenu + vocalItems + vocalInput + guidance  Singers: Chris Hay Lee Hardy Paul Sheerin Michael Moore Jordan Tait Andy Jackson Stevie McManus

Table 3-4. Correspondence between popular widgets and different types of CIOs

3.2.4 Concrete User Interface Relationship semantics

Concrete User Interface Relationships (Figure 3-12) are relationships mapping two or more CIOs. CUI Relationships always have at least one source object and at least one target object. There are three types of relationships:

1. *GraphicalRelationship:* is a relationship mapping two or more graphicalCIOs. It is sub-divided in:
    - *GraphicalTransition:* is a relationship mapping one or several GCs by specifying a navigation structure among the different containers populating a cuiModel. According to [USIX05] the *transitionType* attribute allows the following values: open, close, minimize, maximize, suspend/resume. In Section 4.3.3.2 we present the *Sub-task navigation*, a design option that specifies the navigation between GCs that support the execution of two different sub-tasks. This navigation implies two simultaneous actions: *deactivate* the GC in which the source sub-task is executed and *activate* the GC in which the target sub-task is executed. We add *activate* and *deactivate* to the already existing set of values in order to offer a more precise identification of transition type between GCs and to support the design options for multimodal UIs. 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:* allows to specify an adjacency constraint between two graphicalCIOs. A graphical modality adjacency may be inferred from the order in which components are place into horizontal box and vertical box. Consequently it is never explicitly stated in the specification.
    - *GraphicalContainment:* allows 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.
    - *GraphicalAlignement:* specifies an alignment constraint between two GICs.
    - *GraphicalEmphasis:* enables to specify that two or more GICs must be differentiated in some way (e.g., with different color attributes).
2. *VocalRelationship:* is a relationship mapping two or more vocalCIOs. It 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). The *transitionEffect* attribute allows a specification of an auditory effect to the transition (e.g., fade-out, fade-in).
    - *VocalAdjacency:* allows specifying 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 or one or more VICs. This relationship is particularly useful for adding or deleting VICs from VCs.
3. *CuiDialogControl:* enables the specification of a dialog control in terms of LOTOS operators between any types of CIOs, be it graphical or vocal.
4. *Synchronization:* is a relationship that synchronizes vocalCIOs with graphicalCIOs in multimodal UIs. The two interaction parts in a multimodal UI specified with UsiXML, the vocal part and the graphical part are syntactically separated one of each other

(Requirement 7. Separation of modalities). The synchronization between them 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 fulfill her name in a text field by employing the vocal modality, the recognized result can be found both in the vocal element and in the text field. If the user is typing the name, the introduced value will update both the text field and the vocal element. There are four cases when the synchronization relationship can be used:

- *Synchronization between 1 VIC and 1 GIC:* the synchronization will be defined directly between the VIC (i.e., the source) and the GIC (i.e., the target). For instance, if we consider the task of fulfilling vocally the name of a person in a text field, the designer has to ensure the synchronization between the vocal input and the text field.
- *Synchronization between 1 VIC and n GICs:* the synchronization will be defined between the VIC (i.e., the source) and the GC (i.e., the target) that embeds the GICs. If the designer of the multimodal UI takes the decision of synchronizing the VIC with the set of n GICs, then he must to embed the last ones into a GC. For instance, we consider the task of fulfilling vocally the date in a form by using three combo boxes (one for day, one for month and one for year). If the designer wants to allow the fulfillment of the task with only one vocal input (e.g., "5th of May 2006") then he has to embed all three combo boxes in the same GC (e.g., a group box) that will be synchronized with the vocal 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 selecting vocally the date in a date picker widget by using three separate vocal inputs (one for day, one for month and one for year), the designer has to embed all three vocal inputs in the same VC (e.g., vocalForm) that will be synchronized with the GIC.
- *Synchronization between n VIC and n GIC:* this situation must be decomposed 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 establish by considering the appearance order of the VICs and GICs.

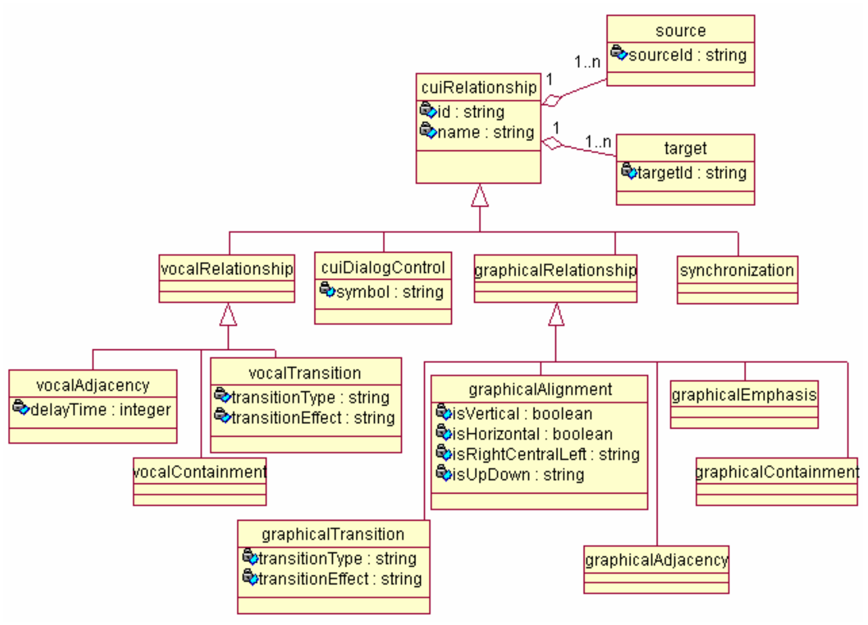


Figure 3-14. Concrete UI Relationships

3.3 Syntax

Syntax is often opposed to semantics, in which case semantics pertains to what something means, while syntax pertains to the formal structure in which something is expressed.

3.3.1 From Semantics to Concrete Syntax

As described in Sections 3.1 and 3.2, the semantics of UsiXML language is encoded as UML Class Diagram. The syntax of UsiXML language has an XML-based format structure. XML stands for eXtensible Markup Language and allows describing sets of data with a tree-like structure. For the definition of valid XML elements, UsiXML considers XML Schemas [W3C01]. Figure 3-15 illustrates the manual transformations (T1) applied in order to produce XML Schemas from UsiXML class diagram concepts. The Instances of UsiXML Class Diagram Concepts, i.e., the UsiXML objects, are transformed into UsiXML specification (T2). Finally, the resultant specification is validated by the corresponding XML Schema.

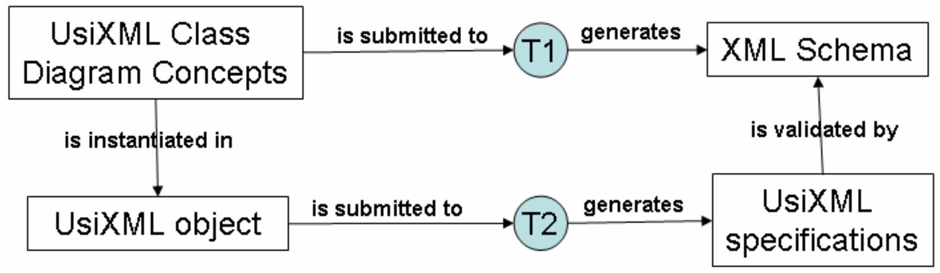
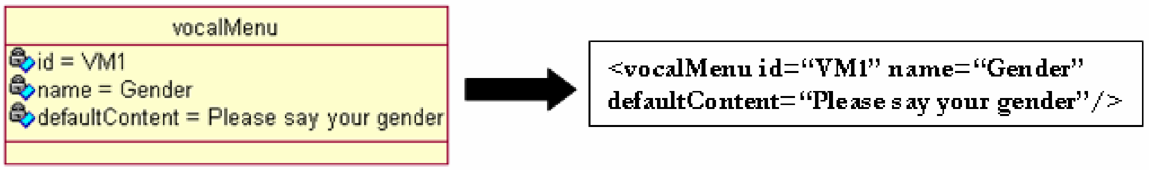


Figure 3-15. Generation of UsiXML specification

In the following we specify a set of UsiXML Class Diagram Concepts that are subject of the transformation T1 illustrated in Figure 3-15. For each transformation an example at of UsiXML objects submitted to transformations T2 is provided:

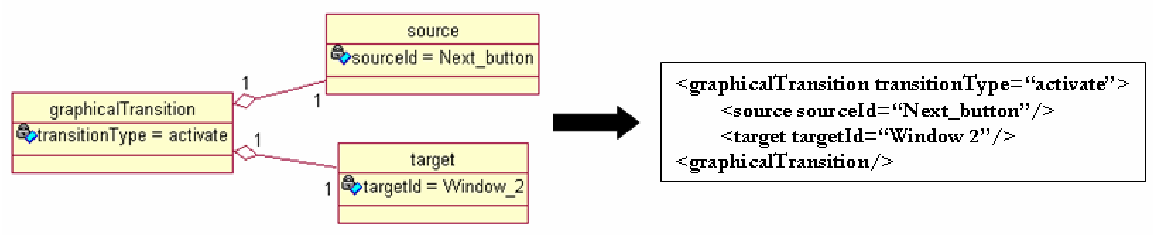
- A *class* becomes an *XML element* and *class attributes* become *XML attributes*. Figure 3-16 exemplifies how an instance of a class is mapped into an XML element with the associated attributes.



```
<vocalMenu id="VM1" name="Gender"
defaultContent="Please say your gender"/>
```

Figure 3-16. Transforming Class to XML element

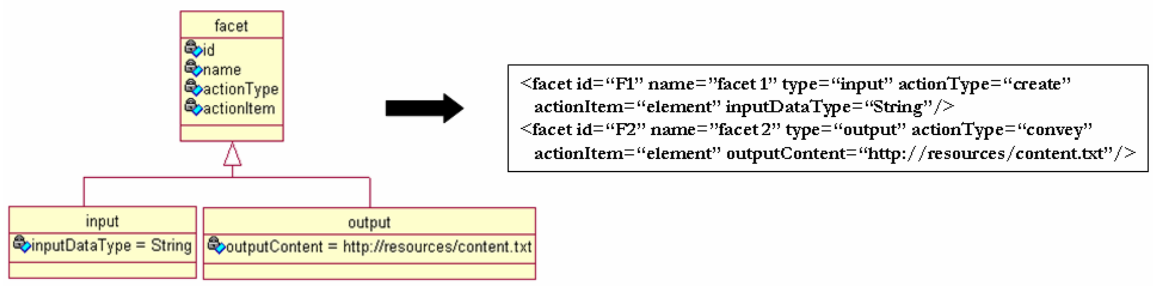
- *UsiXML source-target relationships* are translated into an *XML element* whose name specifies the name of the relationship. The created element embeds a *source* and a *target element* that designates the source and the target of the relationship, respectively. Figure 3-17 exemplifies how a *graphicalTransition* relationship between two elements (i.e., a source represented by a *button* and a target represented by a *window*) is transformed into UsiXML specification.



```
<graphicalTransition transitionType="activate">
      <source sourceId="Next_button"/>
      <target targetId="Window 2"/>
<graphicalTransition/>
```

Figure 3-17. Transformation of UsiXML source-target relationship

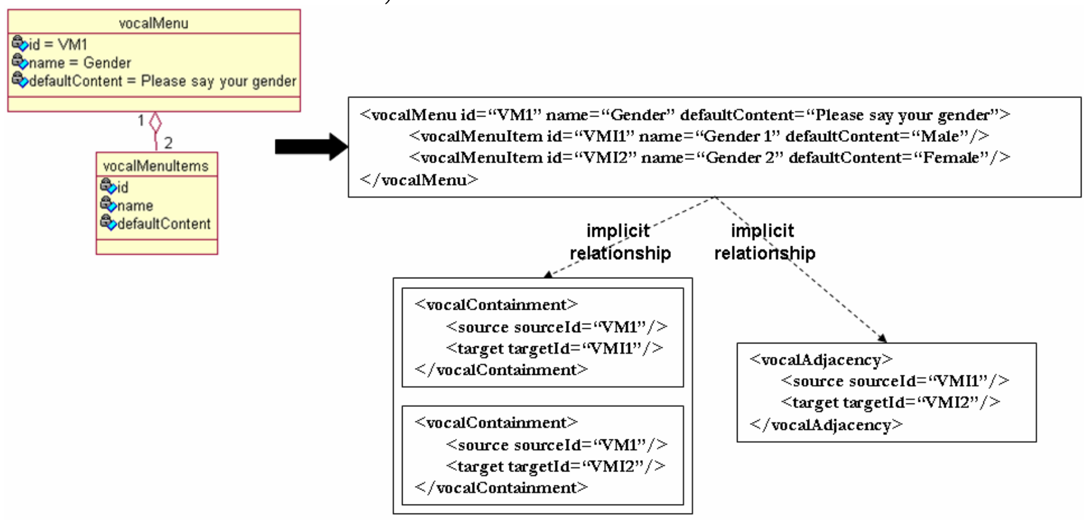
- *Inheritance relationship* is transformed into an *XML element* whose name is the name of the superclass. The name of the subclass is specified by the *type* attribute value of the superclass and the attributes of the subclass become XML attributes of the created element. Figure 3-18 presents two instances corresponding to two different classes (i.e., *input* and *output*) that inherit attributes from the same superclass (i.e., *facet*). For each instance 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.



```
<facet id="F1" name="facet 1" type="input" actionType="create"
  actionItem="element" inputDataType="String"/>
<facet id="F2" name="facet 2" type="output" actionType="convey"
  actionItem="element" outputContent="http://resources/content.txt"/>
```

Figure 3-18. Transforming inheritance relationship into an XML structure

- *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 3-16. The XML element generated from the client class embeds the XML element generated from the supplier class. Figure 3-19 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 implicitly relationships between objects. For instance, in Figure 3-19 the structure of UsiXML syntax allows to infer two *vocalContainment* relationships: the *vocalMenu VM1* embeds the *vocalMenuItems VMI1* and VMI2, repsectively. Moreover, an *adjacency* relationship may be deduced from the order of XML elements: there is a *vocalAdjacency* between *vocalMenuItem VMI1* and *vocalMenuItem VMI2* inferred from the ordering of declaration of these elements within the *vocalMenu VM1*).



```
<vocalMenu id="VM1" name="Gender" defaultContent="Please say your gender">
      <vocalMenuItem id="VMI1" name="Gender 1" defaultContent="Male"/>
      <vocalMenuItem id="VMI2" name="Gender 2" defaultContent="Female"/>
</vocalMenu>
```

```
<vocalContainment>
      <source sourceId="VM1"/>
      <target targetId="VMI1"/>
</vocalContainment>
```

```
<vocalContainment>
      <source sourceId="VM1"/>
      <target targetId="VMI2"/>
</vocalContainment>
```

```
<vocalAdjacency>
      <source sourceId="VMI1"/>
      <target targetId="VMI2"/>
</vocalAdjacency>
```

Figure 3-19. Transforming aggregation relationship into an XML structure

3.3.2 Concrete Syntax of Interaction Objects

In this section we provide the UsiXML syntax for a series of widgets expressed in three different types of UIs: graphical, vocal and multimodal. For multimodal UIs we provide also the syntax for the *synchronization* relationship (see Section 3.2.4). Our methodology aims at covering the CARE properties (Requirement 3. CARE properties support for input modalities and Requirement 4. CARE properties support for output modalities). Due to the fact that in this dissertation we target X+V applications, only *Assignment,* E*quivalence* and *Redundancy* are supported. The *Complementarity* property requires the system to perform data fusssion for input modalities and data fission for output modalities. Neither data fusion nor data fission are currently supported by X+V browsers. The current work takes into account only *graphical* and *vocal* modalities. Thus, we distinguish the following types of interaction:

- *Graphical interaction:* no synchronization is required as there are only graphical components involved
- *Vocal interaction:* no synchronization is required as there are only vocal components involved
- *Multimodal interaction with graphical assignment for input:* no synchronization is required as there are not any vocal input components involved
- *Multimodal interaction with vocal assignment for input:* synchronization between the vocal input and the associated graphical component is required (*isEnabled*

attribute of the graphical component is set to *false* in order to block the graphical interaction)

- *Multimodal interaction with equivalent input (graphical or vocal):* synchronization between the vocal input and the associated graphical component is required (*isEnabled* attribute of the graphical component is set to *true* in order to allow the graphical interaction)
- *Multimodal interaction with redundant input (graphical and vocal):* no synchronization is required as the two inputs (i.e., the graphical and vocal) are gatherd by two different chanels
- *Multimodal interaction with redundant output (graphical and vocal):* no synchronization is required as the two outputs (i.e., the graphical and vocal) are provided by two different chanels.

Our specification is based on Table 3-4. For each considered set of widgets we provide the corresponding specification for some of the interaction types identified above. Here we exemplify the Label + Text field widgets, while the specification for the rest of the widges can be found in Annex A.

LABEL + TEXT FIELD:

- **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="§var".../>
</box>
```

- **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" currentValue="§var".../>
   <vocalFeedback id="F1" name="Feedback 1" defaultContent="Your name is §var".../>
</vocalForm>
```

- **Multimodal interaction with graphical assignment for input and redundant output:**

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

<box id="b1" name="Box 1"...>
   <outputText id="OT1" name="Output 1" defaultContent="Name".../>
   <inputText id="IT1" name="Input 1" isEditable="true" currentValue="§var”.../>
   <imageComponent id="IC3" name="speaker_icon" defaultContent="speaker.jpg".../>
</box>
```

- **Multimodal interaction with vocal assignment for input and redundant output:**

```
<vocalForm id="VF1" name="Form 1"...>
   <vocalPrompt id="VP1" name="Prompt 1" defaultContent="Please say your name".../>
   <vocalInput id="VI1" name="Input 1" currentValue="§var".../>
   <vocalFeedback id="F1" name="Feedback 1" defaultContent="Your name is §var".../>
</vocalForm>

<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".../>
    <imageComponent id="IC3" name="speaker_icon" defaultContent="speaker.jpg".../>
</box>

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

- **Multimodal interaction with equivalent input and redundant output:**

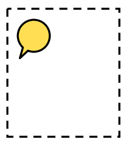
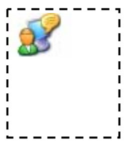
```
<vocalForm id="VF1" name="Form 1"...>
    <vocalPrompt id="VP1" name="Prompt 1" defaultContent="Please say your name".../>
    <vocalInput id="VI1" name="Input 1" currentValue="§var".../>
    <vocalFeedback id="F1" name="Feedback 1" defaultContent="Your name is §var1".../>
</vocalForm>

<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="IC2" name="keyboard_icon" defaultContent="keyboard.jpg".../>
    <inputText id="IT1" name="Input 1" isEditable="true" currentValue="§var1".../>
    <imageComponent id="IC3" name="speaker_icon" defaultContent="speaker.jpg".../>
</box>

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

3.4 Stylistics

The objective of stylistics is to provide a representation of a set of defined objects in order to facilitate their understanding and manipulation in tools. The representation can be of different types (e.g., graphical, textual). The current section provides a graphical representation for the vocal concrete interaction objects defined in Section 3.2.2.

vocalCIO	**Graphical representation**
vocalGroup	
vocalForm	
vocalMenu with *vocalMenuItems*	

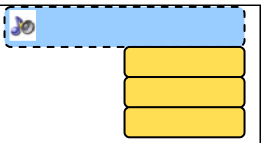
	
vocalConfirmation	
vocalFeedback	
vocalPrompt	
vocalInput	
vocalNavigation	
break	
exit	

Table 3-5. Stylistics for vocal concrete interaction objects

3.5 Conclusions

The current chapter detailed the concepts of the framework selected and extended for the development of multimodal web applications. The most important introduced concepts refer to the vocal and multimodal concrete interaction objects (vocalGroup, vocalForm, vocalMenu, vocalInput, vocalOutput, multimodal texfield, multimodal group of radio buttons, etc.) for which we provided the semantics and syntax. In order to synchronize the vocal and the graphical concrete interaction objects which are involved in multimodal interactions, the *synchronization* relationship was introduced. In Chapter 4 we present in detail a transformational approach based on design options for producing multimodal applications.

CHAPTER 4 A TRANSFORMATIONAL METHOD FOR PRODUCING MULTIMODAL WEB USER INTERFACES

After introducing in Chapter 3 the concepts of our multimodal framework, the Chapter 4 presents the transformational method for producing multimodal web applications. Section 4.2 describes the details concerning the specification of transformations. Section 4.3 introduces a design space composed of design options for graphical and multimodal web user interfaces. Based on these design options the 4 steps of the transformational approach are detailed in Section 4.4. Finally, we present in Section 4.5 the tool support for the transformational method.

4.1 Specification of transformation

Model-to-model transformations 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 Model Driven Architecture (MDA) [Mill03]. Several techniques have been surveyed in the literature [Czar03], while in [Limb04] the most relevant ones are presented along with their shortcomings:

- *Imperative languages:* text-processing languages that perform 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 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. Transformations are grouped in transformation tasks, which are themselves 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, CWM does not provide us with a predefined language to specify the way elements are transformed one to another.

In the context of this dissertation, model-based approaches for the development of UIs, in general, and for MWUIs, in particular, involve a transformational approach which consists of a successive application of transformations over initial abstract models that allow refining them into more concrete models, until the *Final User Interface* is achieved [Stan05].

The current work considers a transformational approach based on *graph transformation rules* in order to progressively move from the uppermost level, the *Task* and *Domain Models,* to *Abstract Model* from which a *Concrete Model* is derived (Figure 4-1).

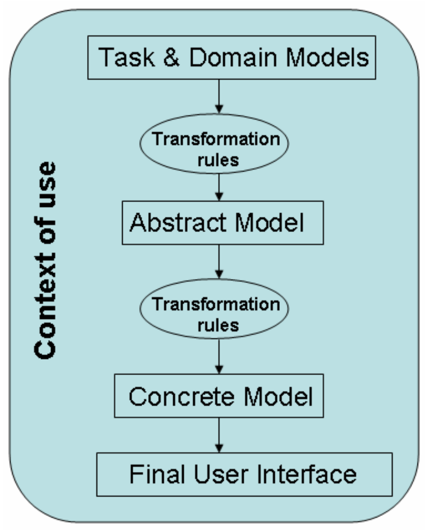


Figure 4-1. Progressive application of Rule – based transformations

In the following we motivate our choice based on [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 4-2 details the features of the *Transformation Rule* approach based on which we emphasize the coverage of our transformational approach.

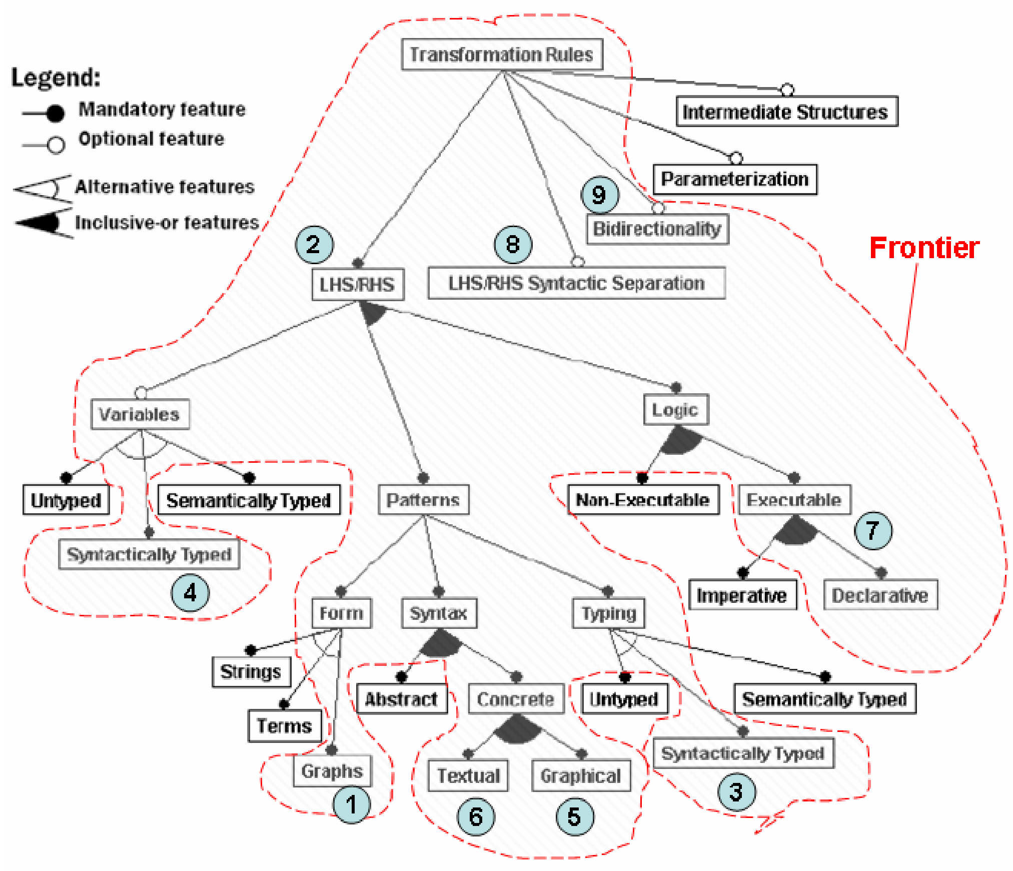


Figure 4-2. Identification of transformation rule approach features

1. Graph-based patterns

To ensure the progressive approach illustrated in Figure 4-1, UsiXML provides a *Transformation Model* (see Section 3.1.8) containing a set of rules that applies successive transformation to an initial representation (Requirement 9. Transformation-based development). Transformations are encoded as *graph transformation rules* performed on the involved models expressed in their graph equivalent (Requirement 12. 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 4-3). The rules in a transformation catalog are structured in *development steps*. For instance, passing from *Task Model* to *Abstract Model* or passing from *Abstract Model* to *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 one (and only one) *transformation system* and a transformation system is realized by a *set of graph transformation rules*.

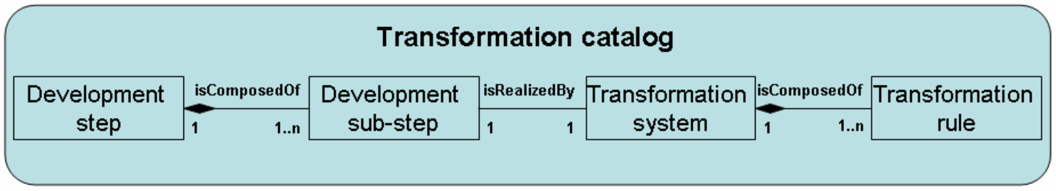


Figure 4-3. The structure of a transformation catalog

2. LHS/RHS

Graph rewriting rules are based on a pattern matching mechanism that selects a sub-graph in a graph structure and applies to this sub-graph any type of transformation (adding, deleting or modifying a node or an edge). A graph rewriting rule is defined as a set of two graphs:

- ✓ *LHS:* is the *Left Hand Side* of the rule. It expresses a graph pattern that, if it matches in 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:* is the *Right Hand Side* of the rule. It is the graph that will replace the LHS in the host graph.
- ✓ *NAC:* is the *Negative Application Condition* of the rule. The graph rewriting rules used in the current dissertation are conditional graph rewriting rules. This component is added in order to 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 4-4 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:

a. Find an occurrence of LHS into G (this occurrence is called a *match*). If several occurrences exist, choose one non-deterministically.
b. Check that NAC does not match into G. If there is a match then skip to onother occurrence of LHS
c. 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.

The application strategy of the transformation systems supposes a sequential execution of rules. Once a rule from a transformation system terminates to be applied, the next rule will be executed and so forth until the last rule terminates. Then the next transformation systems from the transformation catalog will be executed sequentially until the last one.

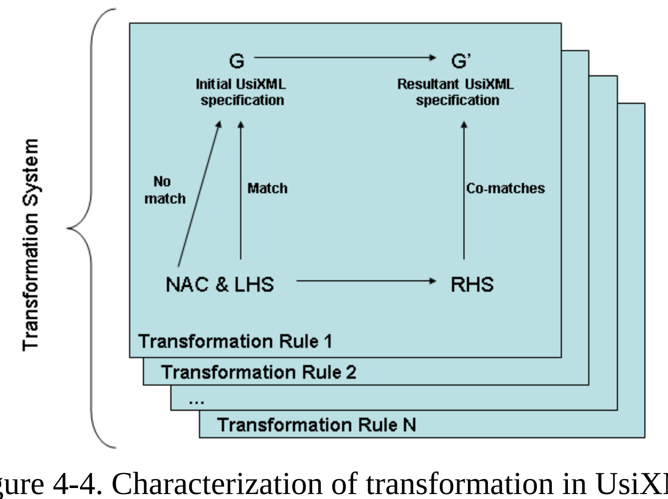


Figure 4-4. Characterization of transformation in UsiXML

3. Syntactically Typed Patterns

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 edges) to the same types indicates a commonality in terms of properties between these nodes (or edges). Figure 4-5 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.

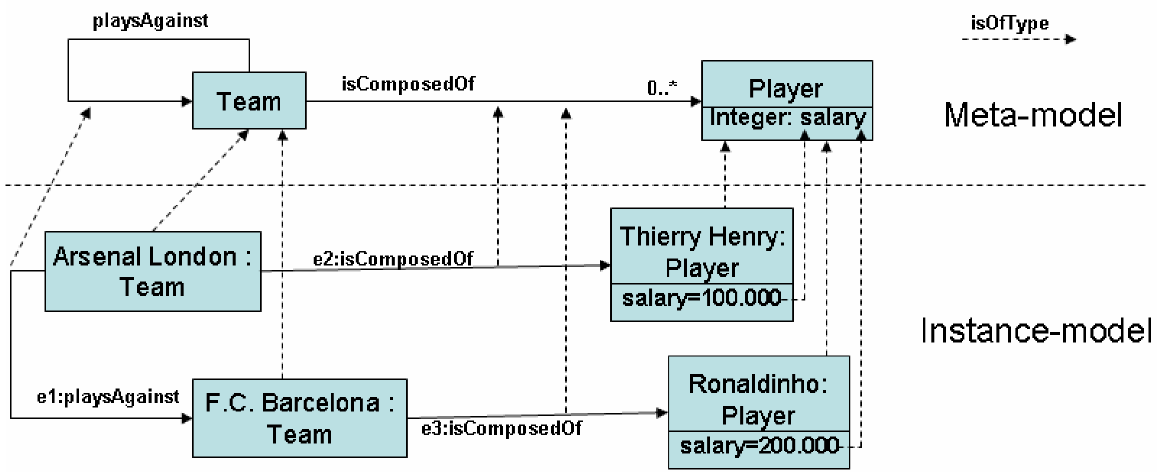


Figure 4-5. Syntactically typed patterns and variables

4. Syntactically Typed Variables

Similar with patterns, syntactically typed variables are variables that are associated with meta-model elements whose instances it can hold. Figure 4-5 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 4-6 illustrates the graphical notations for: nodes, edges, node and edge types and node and edge attribute (variable) values.

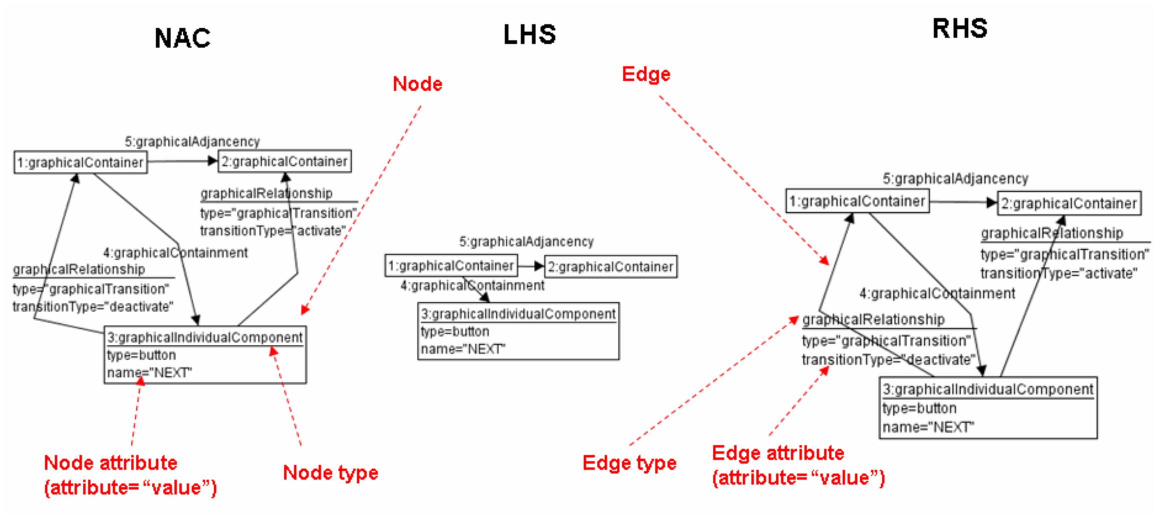


Figure 4-6. Graphical concrete syntax of the patterns

Figure 4-7 shows one of the rules that can be applied to pass from the *Task Model* to the *Abstract Model*. The rule can be interpreted as follows: 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 (RHS). In order to map the corresponding elements of the NAC, LHS and RHS components of a rule, the graph formalism uses digits in front of mapped nodes and edges (e.g., task 1 described in LHS corresponds to task 1 from NAC and from RHS).

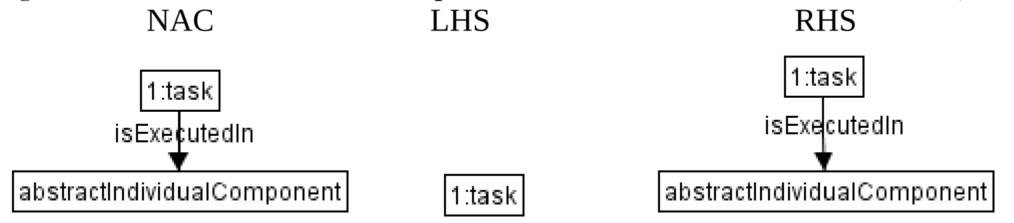


Figure 4-7. From Task Model to Abstract Model

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 4-8 offers an example of the equivalent textual syntax of the rule illustrated in Figure 4-7.

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

```
            <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 4-8. 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 4-8) specifically marks the LHS, RHS and NAC parts.

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 where designed to move forward and backward between different UI models.

4.2 Design space for web user interfaces

The capabilities of multimodal applications running on the web are well delineated since they are mainly constrained by what their underlying standard markup language offers, as opposed to hand-made multimodal applications. As the experience in developing such multimodal web applications is growing, the need arises to identify and define major design options of such application 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 User Interface 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 favor of a design option. For instance, the design option “label location” in a UI layout could be set to “left-aligned”, “centered”, or “right-aligned” depending on the layout type to be adopted for the end user. When a particular design option value is assigned to a design option, it is considered that a *design decision* is taken among the stakeholders representing the various interests in the development life cycle (e.g., the end users, the marketing, and the development team). The combination of design options forms a *design space* (Requirement 5. Approach based on design space). 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 a *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. In this case, the design space becomes a snowflake model.

Design options often involve various stakeholders representing different human populations with their own preferences and interests. Consequently, design decisions often result from a process where the various design options are gathered, examined, and ranked until an agreement is reach among stakeholders. This decision process is intrinsically led by consensus since stakeholders' interests may diverge and by trade-off between multiple criteria, which are themselves potentially contradictory.

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 incorporating 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 "label location" will be transformed into widgets' locations satisfying the corresponding constraints.
- The adoption of a design space supports the tractability of more complex design problems or for a class of related problems.

On the other hand, design options also suffer from some shortcomings: 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, or imagination, or background
- Their decision is not always clear and when they are decided, they may violate some other usability principle or guideline. 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 always represents a restrictions of the complete design space, the problem being to identify the relevant ones and leaving out the too detailed ones 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. Not all the possible values of a design option may be equal in implementation cost.

We believe that it is important to define such a design space for web applications because of several reasons: the languages in which they are implemented (e.g., XHTML, VoiceXML, X+V) restrict the amount of possible interfaces to obtain and directly set the CARE properties to assignment and equivalence. In addition, the interaction styles [Beau00] supported by these languages make them appropriate for certain types of applications (e.g., information systems), but totally inadequate for other types (e.g., air traffic control) [Macl89]. Multimodal web applications typically combine three interaction modalities: graphical (e.g., a XHMTL web page in a web browser), vocal (e.g., a VoiceXML application through a multimodal web browser), tactile (e.g., a X+V application running on an interactive kiosk equipped with a tactile screen). Any of these modalities could be combined together, thus multiplying the combination of design options which are specific to each modality and complexifying the entire design space. Sometimes, a design option which was estimated relevant for a particular modality (say the graphical channel) may become totally irrelevant when this modality is combined with another one (say with the vocal channel). The fusion/fission mechanism is generally the one implemented in the browser with which the multimodal web application is run. 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 a multimodal web application. This provision avoids designers to replicate the identification and definition of these design options, while leaving them free to consider other options or to overwrite existing ones.

4.2.1 Design options for graphical web user interfaces

We have identified a number of five design options for graphical web user interfaces. These design options are illustrated in Figure 4-9.

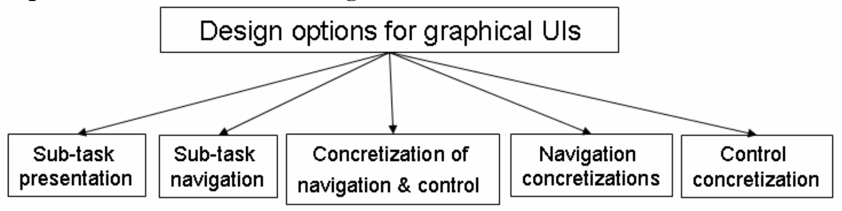

Design options for graphical UIs
Sub-task presentation
Sub-task navigation
Concretization of navigation & control
Navigation concretizations
Control concretization

Figure 4-9. Design options for graphical web user interfaces

Sub-task presentation. Specifies the appearance of each sub-task in the final user interface. The possible values are illustrated in Figure 4-10. The presentation of each sub-task can be either separated or combined. Separated presentation identifies the situation when each sub-task is represented in different containers (e.g., different windows), while the combined value identifies the situation when all sub-tasks are presented in the same container. In the last case three different types of combinations are possible:

- *One by one*: only one sub-task is presented at a time (e.g., in combined box, in tabbed dialog box, in float window)
- *Many at once*: multiple sub-tasks are presented in the same time (e.g., in float window)
- *All in one*: all sub-tasks are presented in the same time (e.g., in areas with separators, in group boxes, in bulleted list, in numbered list).

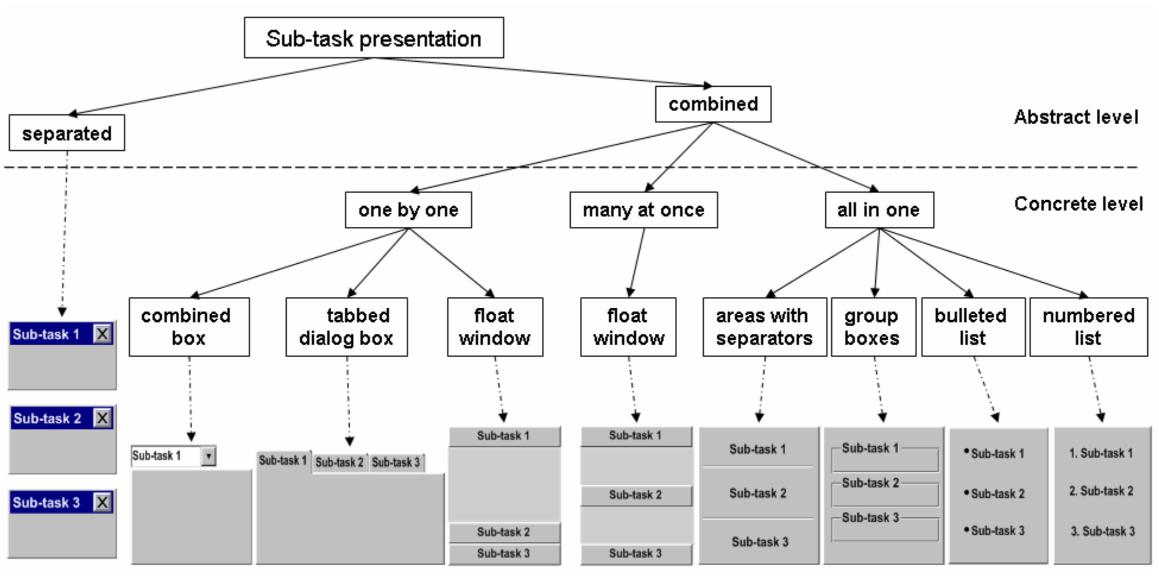


Figure 4-10. Final representation of sub-task presentation design option

Sub-task navigation. Is an extension of the notation introduced in [Vand03]. It specifies the way in which the navigation between the sub-tasks is ensured. Figure 4-11 illustrates the two possible values: *sequential* or *asynchronous*. The sequential navigation, also called synchronous, restricts the transfer of the dialog control only to a neighbor presented sub-task. The asynchronous type offers more flexibility in transferring the dialog control by eliminating the above restriction and allowing a transfer from any source sub-task to any target sub-task. Passing the dialog control from a source sub-task to a target sub-task implies two simultaneous actions (see Section 3.2.4): *deactivate* the container in which the source sub-task is executed (represented here with a yellow bulb) and *activate* the container in which the target sub-task is executed (represented here with a red bulb).

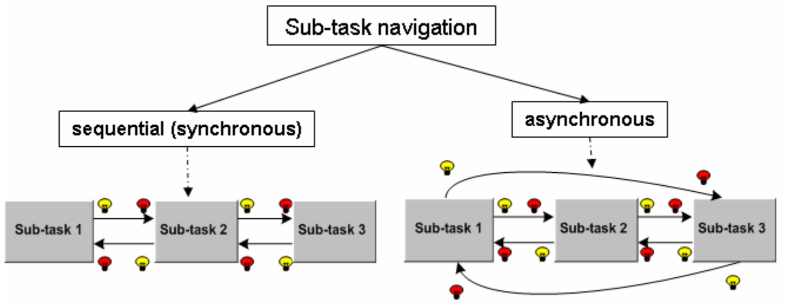


Figure 4-11. Types of navigation between sub-tasks

Concretization of navigation and control. Is a design option that specifies if the navigation and control are ensured by the same object. In Figure 4-12, the *separated* value identifies the situation in which the control and the navigation between the sub-tasks are attached to different objects/logically grouped set of objects. When the same object ensures simultaneously the navigation and the control, the value of the design option is *combined*.

Navigation concretization. Identifies the *placement* and the *cardinality* of the navigation objects/logically grouped set of objects that ensure the navigation. Figure 4-13 illustrates the different values of the design option. There are two *types of placement* for the navigation objects:

- *Local placement:* specifies the existence of a navigation object attached to each presented sub-tasks
- *Global placement:* a general object ensures the whole navigation between the sub-tasks.

The *cardinality* specifies the number of objects that ensures the navigation. We have identified two *types of cardinality*:

- *Simple:* the navigation is ensured by a single object (e.g., an *OK* button) or a single logically grouped set of objects (e.g., the *(NEXT, PREVIOUS)* group of buttons)
- *Multiple:* the navigation is ensured simultaneously by two or more objects/logically grouped set of objects (e.g., the group of tab items in a tabbed dialog box and the *(NEXT, PREVIOUS)* group of buttons).

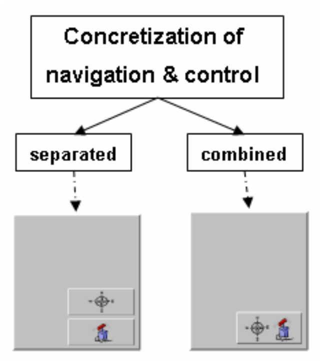


Figure 4-12. Navigation and control concretization

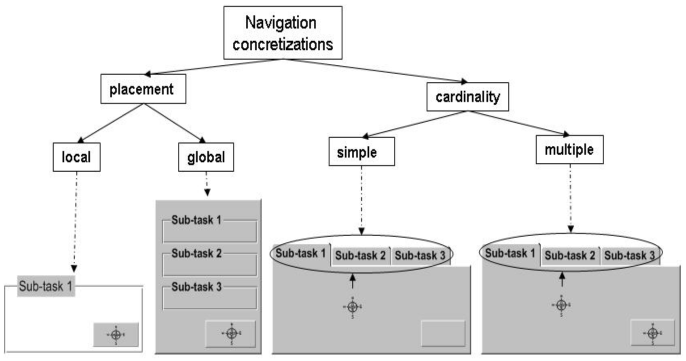


Figure 4-13. Types of navigation

Control concretization. Identifies the *placement* and *cardinality* of the control objects. Figure 4-14 illustrates the different values of these design options. There are two *types of placement* of the control objects:

- *Local placement:* specifies the existence of a control object attached to each presented sub-tasks
- *Global placement:* a general object ensures the control for each sub-task.

The *cardinality* specifies the number of objects that ensure the control. We have identified two *types of cardinality*:

- *Simple:* the navigation is ensured by a single object (e.g., an *OK* button) or single logically grouped set of objects (e.g., the group of tab items in a tabbed dialog box)
- *Multiple:* the navigation is ensured simultaneously by two or more objects/logically grouped set of objects (e.g., the group of tab items in a tabbed dialog box and the *(NEXT, PREVIOUS)* group of buttons).

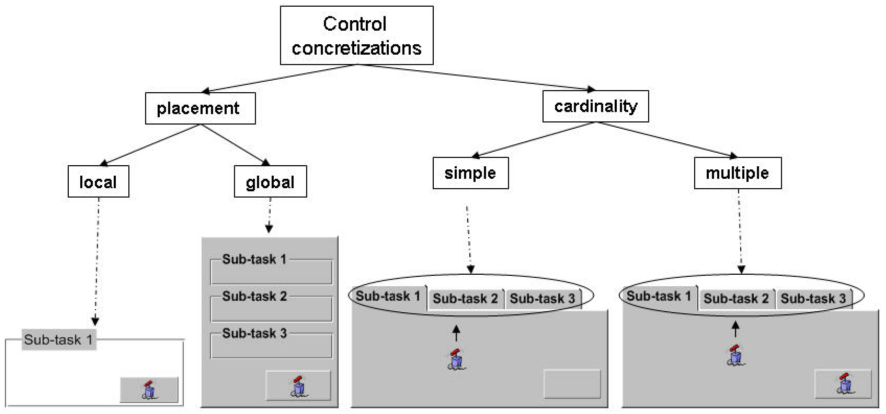


Figure 4-14. Types of control concretization

4.2.2 Design options for vocal web user interfaces

We have identified a number of five design options for vocal user interfaces. These design options are illustrated in Figure 4-15.

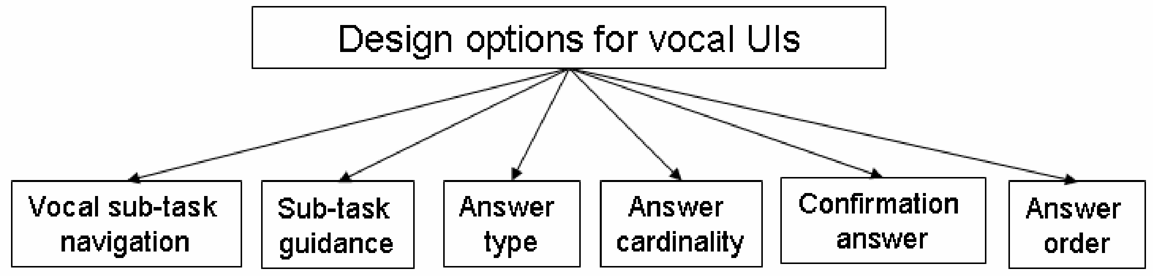


Figure 4-15. Design options for vocal user interfaces

Vocal sub-task navigation. Specifies the way in which the navigation between the vocal sub-tasks is ensured. This design option has two possible values:

- *Sequential:* the order of the sub-task fulfillment is decided be the system. An example of sequential navigation is presented in the following:
  System: "What is your zip code?"
  User: "1348"
  System: "Please say your gender."
  User: "Female"
- *Asynchrounous:* the order of the sub-task fulfillment is specified by the end-user. An example of asynchrounous navigation is offered below:
  System: "What information would you like to input?"
  User: "Gender"
  System: "Please say your gender."
  User: "Female"

Sub-task guidance. Specifies if the end-user in guided with the possible answers he/she might input. The possible values are:

- *With guidance:* the system provides the possible answers to the end-user. An example is offered in the following:
  System: "Select the car color. Choose between green, red and black."
  User: "Red"

- *Without guidance:* no possible answer is provided. This situation appears especially when the answers are predictable. For instance:
  System: "What is your gender?"
  User: "Male".

Answer type. Specifies if there are any ambiguities in the users's answer. The possible values are:

- *With disambiguities:* the user's answer is not detailed enougth. For instance:
  System: "What is your weight?"
  User: "85"
- *Without disabiguities:* the users's answer is very precis. For instance:
  System: "What is your weight?"
  User: "85 kilograms"

Answer cardinality. Specifies the number of items that are composing an end-user's answer. The possible values are:

- *Simple:* the end-user can select only one item in an answer. For instance:
  System: "Which are your hobbies?"
  User: "Sport".
- *Multiple:* the end-user can select two or more items in an answer. For instance:
  System: "Which are your hobbies?"
  User: "Sport and music."

Confirmation answer. Specifies if the system requires a supplementary confirmation for a user's answer. The possible values are:

- *With confirmation:* the confirmation is required so as to clarify the answer. For instance:
  System: "What is your weight?"
  User: "85"
  System: "85 kilograms?"
  User: "Yes".
- *Without confirmation:* the user's answer is clear enougth so that no confirmation is required. If we consider the example below, the dialog will be as follows:
  System: "What is your weight?"
  User: "85"

Answer order. Specifies the order in which the user's answers will be treated by the system. The possible values are:

- *Order dependent:* the user can input the answers in a predefined order and the system will process them in the order that they were inputed. For instance:
  System: "What are your gender and age?"
  User: "I am male and I am 24."
- *Order independent:* the user has the flexibility of inputing the answers in any order and the system has to map the answers to the correct location. For instance:
  System: "What are your gender and age?"
  User: "I am 24 and I am male."

System: "Your gender is male and you are 24 years old."

4.2.3 Design options for multimodal web user interfaces

In order to facilitate the development process of multimodal web applications we introduce a set of four design options illustrated in Figure 4-16.

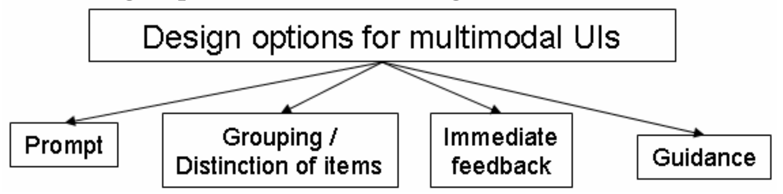

Design options for multimodal UIs
Prompt
Grouping / Distinction of items
Immediate feedback
Guidance

Figure 4-16. Design options for multimodal user interfaces

These design options take into consideration the ergonomic criteria for the evaluation of human-computer interfaces presented in [Bast93]] and adapt them for the development of MWUIs. For simplification, we consider here only two interaction modalities, graphical and vocal, but other modalities can be taken into account later. For each design option we provide a definition and we identify a list of possible values, associating in the same time the corresponding CARE properties described in [Cout95].

Prompting. Refers to the interaction channels available in order to lead the users to take specific actions whether it is data entry or other tasks. The possible values are: *Vocal* (assignment), *Graphical* (assignment), *Multimodal* (equivalence, complementarity or redundancy).

Grouping/Distinction of Items. Concerns the organization of information items in relation to one another. This criterion takes into account the topology (location) and some structural characteristics (format) in order to indicate the relationships between the various items rendered, to indicate whether or not they belong to a given class, or else to indicate differences between classes. This is further decomposed into:

- *Input*: any information input from the user to the system. The possible values are: *Vocal* (assignment), *Graphical* (assignment), *Multimodal* (equivalence).
- *Output*: any information output from the system to the user. The possible values are: *Vocal* (assignment), *Graphical* (assignment), *Multimodal* (equivalence, complementarity or redundancy).

Immediate Feedback. The feedback is necessary for users to interpret the response of the system to their actions [Cole85]. These actions may be simple keyed entries or more complex transactions such as stacked commands. In all cases computer responses must be provided, they should be fast, with appropriate and consistent timing for any transaction. The possible values are: *Vocal* (assignment), *Graphical* (assignment), *Multimodal* (equivalence, complementarity or redundancy).

Guidance. Refers to the means available to advise, orient, inform, instruct, and guide the users throughout their interactions with a computer (messages, alarms, labels, icons, etc.). We offer a more precise level of detail corresponding to the two possible types of interaction considered in this section (i.e., graphical and vocal). Thus, the graphical guidance is sub-divided into *textual* and *iconic*, while the guidance for the vocal interaction can be *acoustic* or based on *speech*. The guidance is further sub-divided in:

- *Guidance for input*: any guidance offered to the user in order to guide him with the input. The possible values are: *Textual* (assignment), *Iconic* (assignment), *Acoustic* (assignment), *Speech* (assignment), or *Multimodal* (by combining the previous values in an equivalent, complementary or redundant way). 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 with the feedback. *Textual* (assignment), *Iconic* (assignment), *Acoustic* (assignment), *Speech* (assignment), or *Multimodal* (by combining the previous values in an equivalent, complementary or redundant way). 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.

4.3 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 4-17.

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:* the step consists in producing one or many AUIs (that are independent of any modality) from the Task and Domain Models previously specified
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 are derived. Each CUI Model can specify graphical, vocal or multimodal UIs.
4. *Step 4: From Concrete User Interface Model to Final User Interface:* from each CUI, a corresponding FUI can be produced by automated generation of code from the models. Thus, for graphical UIs we generate XHTML code, for vocal UIs we produce VoiceXML code, while multimodal UIs are specified using X+V code.

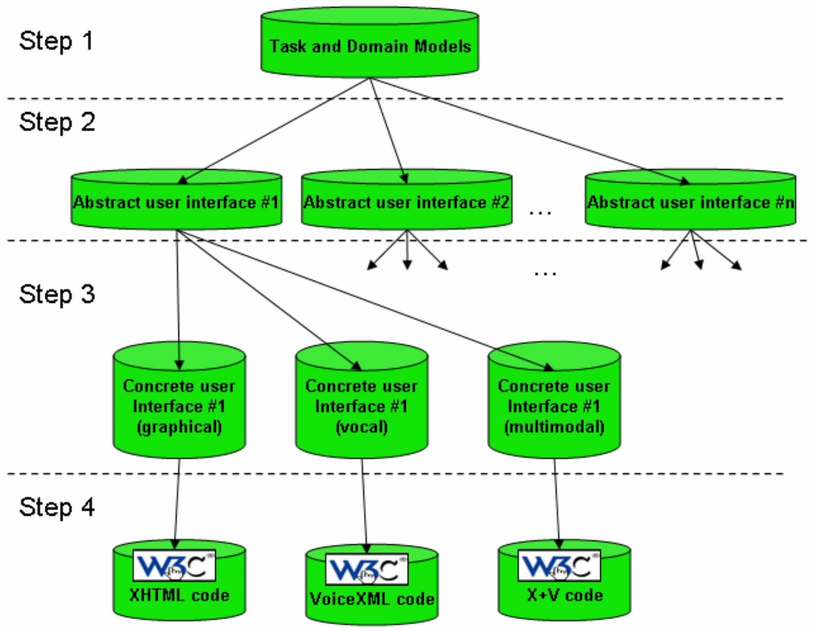

Step 1
Task and Domain Models
Step 2
Abstract user interface #1
Abstract user interface #2
…
Abstract user interface #n
Step 3
…
Concrete user Interface #1 (graphical)
Concrete user Interface #1 (vocal)
Concrete user Interface #1 (multimodal)
Step 4
W3C XHTML code
W3C VoiceXML code
W3C X+V code

Figure 4-17. General development scenario of UIs

In [Limb04b] steps 2 and 3 are further decomposed into sub-steps which consist of transformation systems applied in order to generate graphical and vocal UIs based on

[USIX05]. In this dissertation we define transformation systems by associating the design options defined in Section 4.2 to the different identified sub-steps of the transformational process (Figure 4-18). Based on models described in [USIX06] we generate, not only graphical and vocal UIs but multimodal ones, too.

The analysis of design options for multimodal UIs has to be achieved from two perspectives:

- Feasibility of code generation: refers to the degree in which our transformational approach allows to generate specifications based on design options
- Usability of UIs: is a quality attribute that assesses how easy UIs are to use when developed based on design options. The analysis of the usability takes into consideration the practicability of a selected set of usability principles applied over the design options, over the rules in which they are concretizes and on the application of the rules over the UsiXML models.

The current dissertation is focused on the feasibility of code generation, while the usability of the UIs is let to be done as an internal validation of the Ph.D. thesis associated to the present dissertation.

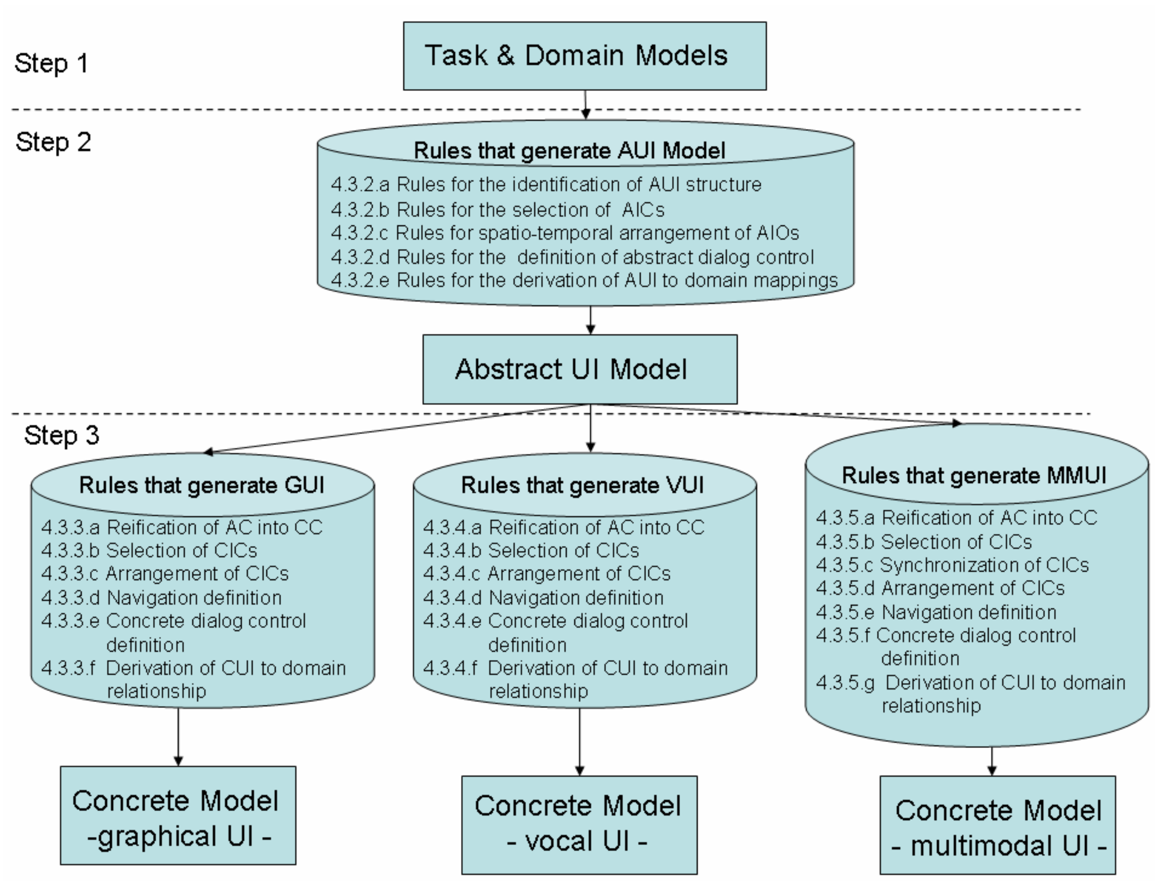

Step 1
Task & Domain Models
Step 2
Rules that generate AUI Model
4.3.2.a Rules for the identification of AUI structure
4.3.2.b Rules for the selection of AICs
4.3.2.c Rules for spatio-temporal arrangement of AIOs
4.3.2.d Rules for the definition of abstract dialog control
4.3.2.e Rules for the derivation of AUI to domain mappings
Abstract UI Model
Step 3
Rules that generate GUI
4.3.3.a Reification of AC into CC
4.3.3.b Selection of CICs
4.3.3.c Arrangement of CICs
4.3.3.d Navigation definition
4.3.3.e Concrete dialog control definition
4.3.3.f Derivation of CUI to domain relationship
Rules that generate VUI
4.3.4.a Reification of AC into CC
4.3.4.b Selection of CICs
4.3.4.c Arrangement of CICs
4.3.4.d Navigation definition
4.3.4.e Concrete dialog control definition
4.3.4.f Derivation of CUI to domain relationship
Rules that generate MMUI
4.3.5.a Reification of AC into CC
4.3.5.b Selection of CICs
4.3.5.c Synchronization of CICs
4.3.5.d Arrangement of CICs
4.3.5.e Navigation definition
4.3.5.f Concrete dialog control definition
4.3.5.g Derivation of CUI to domain relationship
Concrete Model -graphical UI -
Concrete Model - vocal UI -
Concrete Model - multimodal UI -

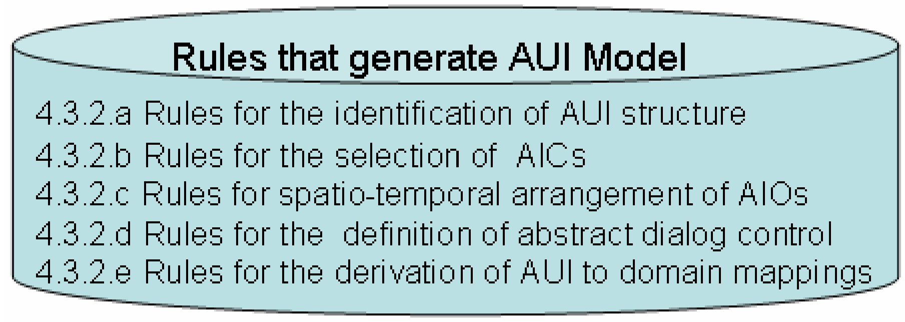
Figure 4-18. Sub-steps of the transformational approach

4.3.1 Step1: The Task and Domain Models

The first development step consists in specifying the Task and Domain Models as a starting point of our transformational approach. The generation of a Task Model (see Section 3.1.1) supposes first the identification of the interactive tasks along with their associated attributes. Further, the relationships between the tasks will be specified. The Domain Model (see Section 3.1.2) consists in 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.

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

The second transformation step involves a transformation system (see Section 4.1) that contains rules applied in order to realize the transition from the Task and Domain Models to Abstract User Interface Model. It consists of the five development sub-steps illustrated in Figure 4-19. The sub-steps are applied following a top-down logical order.


Rules that generate AUI Model

4.3.2.a Rules for the identification of AUI structure
4.3.2.b Rules for the selection of AICs
4.3.2.c Rules for spatio-temporal arrangement of AIOs
4.3.2.d Rules for the definition of abstract dialog control
4.3.2.e Rules for the derivation of AUI to domain mappings

Figure 4-19.Development sub-steps for Step 2: From Task and Domain to AUI

4.3.2.a Rules for the identification of AUI structure

This sub-step consists in defining groups of AIOs that correspond to groups of tasks tightly coupled together. For instance, child of the same task can be considered as a group of tightly coupled tasks.

In order to identify the AUI structure the designer takes into consideration three design options identified in Section 4.2.1:

- Sub-task presentation
- Sub-task navigation
- Navigation concretization
- Control concretization

Sub-task presentation. For the identification of the ACs the designer is resorting to the *sub-task presentation* design option. As in the current step we generate a specification that is independent of the modality, we consider only the values of the design options that can be found at the Abstract level in the Figure 4-10. There are two possible options: the sub-tasks of the same task can be presented separated (Figure 4-20) or combined into the same container (Figure 4-21).

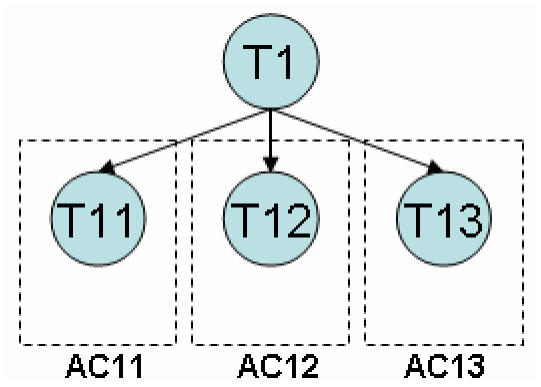


Figure 4-20. Separated sub-task presentation

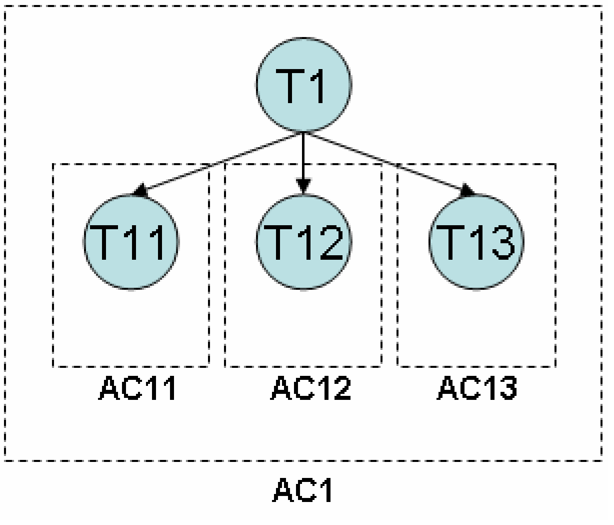


Figure 4-21. Combined sub-task presentation

If the designer's choice is for separated presentation of the sub-tasks the transformation rule illustrated in Figure 4-22 can be applied. An AC is generated for each sub-task of the same task (AC11, AC12 and AC13).

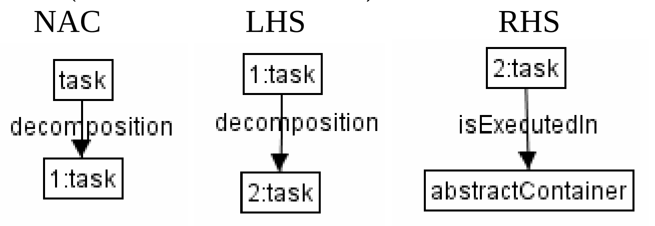


Figure 4-22. Generate abstract containers for each sub-task of the same task

For the second possible choice where the presentation of the sub-tasks is combined two rules are applied. Figures 4-23 and 4-24 show the rules applied in order to create ACs (AC11, AC12 and AC13) for each sub-task of the same task. The sub-tasks will be combined into the same AC (AC1) in which the father task will be executed.

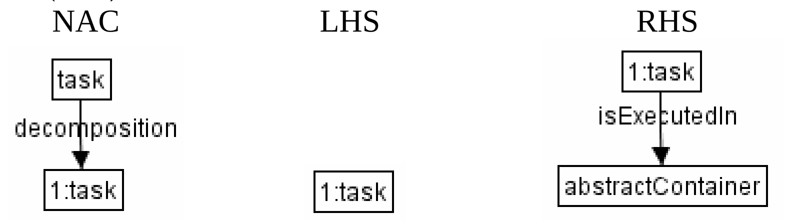


Figure 4-23. Generate an abstract container for the father task

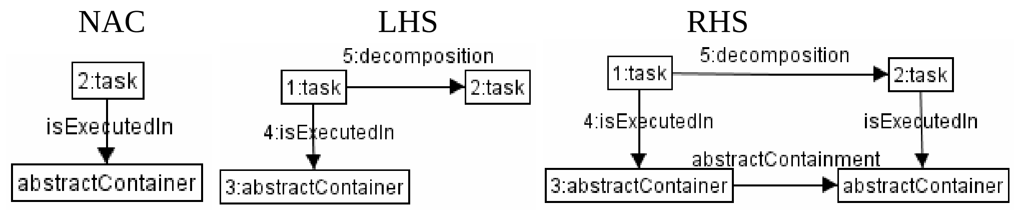
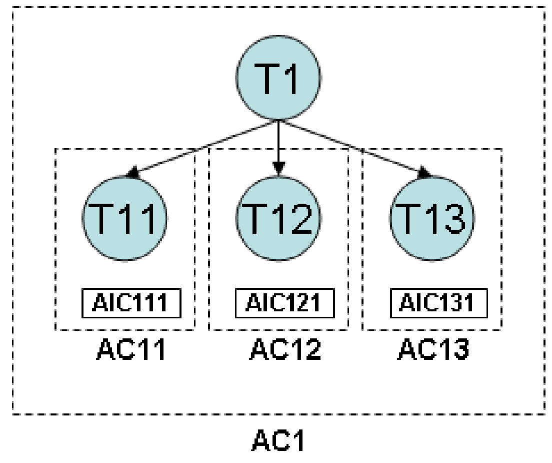


Figure 4-24. Generate abstract containers for each sub-task of the father task

Sub-task navigation. At the Abstract level the designer considers only the generation of AIC that will ensure the navigation between sub-tasks. For instance, if we consider the situation described in Figure 4-25 where each sub-task is executed into a separated AC, rule described in Figure 4-27 generates AICs (AIC111, AIC121, and AIC 131) that will ensure the navigation between these ACs.

Navigation concretization. In order to ensure the placement and the cardinality of AICs, the designer considers the navigation concretization design option. From the placement point of view there are two possible values: local placement (Figure 4-25) of the AICs or global placement (Figure 4-26) of the AIC.



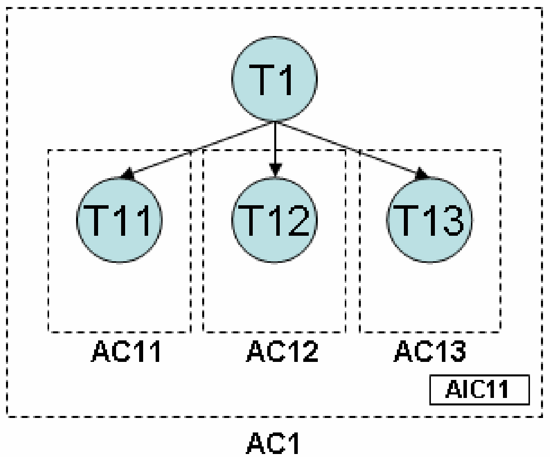


Figure 4-25. Local placement of navigation Figure 4-26. Global placement of navigation

The rules that ensure the *Sub-task navigation* design option (i.e., the generation of AICs) are expressing simultaneously the placement of the AICs. If the designers' decision is for an UI where the navigation objects are locally placed, rule illustrated in Figure 4-27 is generating AICs that are embedded in a corresponding AC (AIC111 embedded into AC11, AIC121 embedded into AC12, etc.). On the contrary, if the designers' decision is for a general object that will ensure the entire navigation between sub-tasks, rule described in Figure 4-28 can be applied to generate a single AIC (AIC11) embedded into the top-most AC (AC1) that ensures the navigation between the embedded ACs (AC11, AC12 and AC13).

NAC LHS RHS

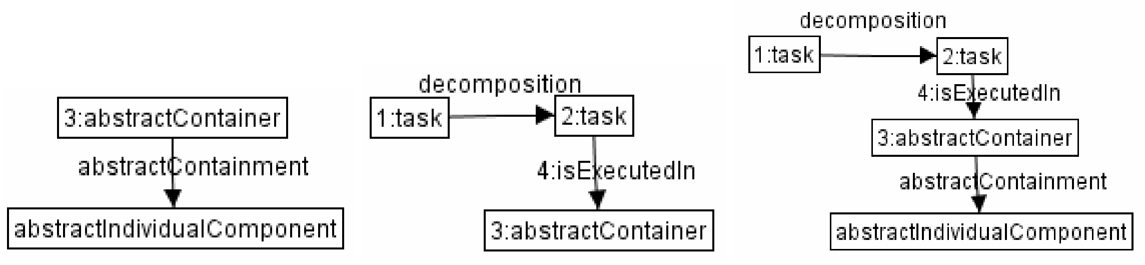


Figure 4-27. Generate local placed AICs that ensure the navigation between sub-tasks

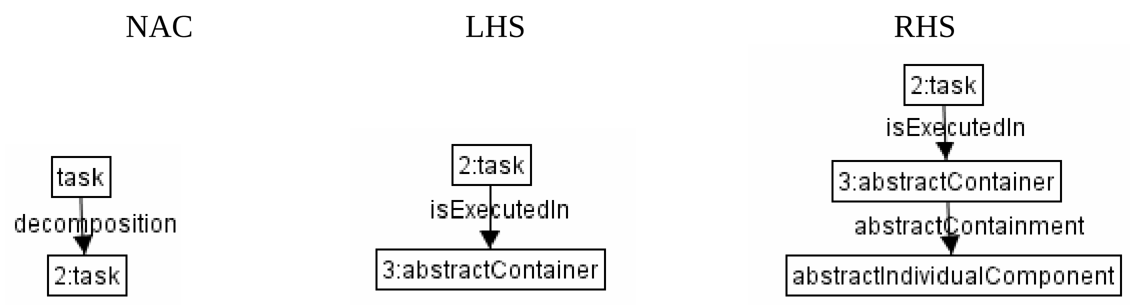


Figure 4-28. Generate one global AIC that ensures the navigation between the sub-tasks

From the cardinality point of view, one can design an UI where the navigation is ensured only by one object (i.e., the cardinality is said to be simple) or simultaneously by two or more objects (i.e., the cardinality is said to be multiple). In both cases described above the cardinality of the navigation is simple as we are generating a single AIC for each sub-task (the case of local placement) and only one AIC for the father task (the case of global placement) that ensures the navigation. In the case of multiple cardinality, other rules have to be developed and applied.

Control concretization. For the identification of the AICs that will ensure the control of data for the sub-tasks mapped into their corresponding AC, the designer takes into consideration the *control concretization* design option. The possible values for this design option are identical with those of navigation concretization. By analogy, the rules applied to generate the AICs that will take in charge the control are the same as those that ensure the navigation concretization.

4.3.2.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 throughout facets, our objective reduces to the selection of the proper facets. In order to achieve this goal we take into consideration the information contained in the Task and Domain Models, in particular the *userAction* and *taskItem* attributes of a task along with the manipulates relationship that adds information on the domain concepts manipulated by the task.

Table 4-1 provides the mappings between task types and AIC facet types. The table does not cover all the possible combinations of actions and items, it takes into account only those that are used in the current dissertation. The left column identifies combinations of values for *userAction* and *taskItem* attributes of a task, while the right column shows the corresponding AIC facets, identifying their values for *actionType* and *actionItem* (see Section 3.1.3).

Task **userAction + 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 4-1. Mappings between tasks types and AIC facets types

In the following we offer three transformational rules in order to exemplify the above described mappings, identifying when possible the corresponding design options. First rule (Figure 4-29) illustrates the generation of 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.

NAC LHS RHS

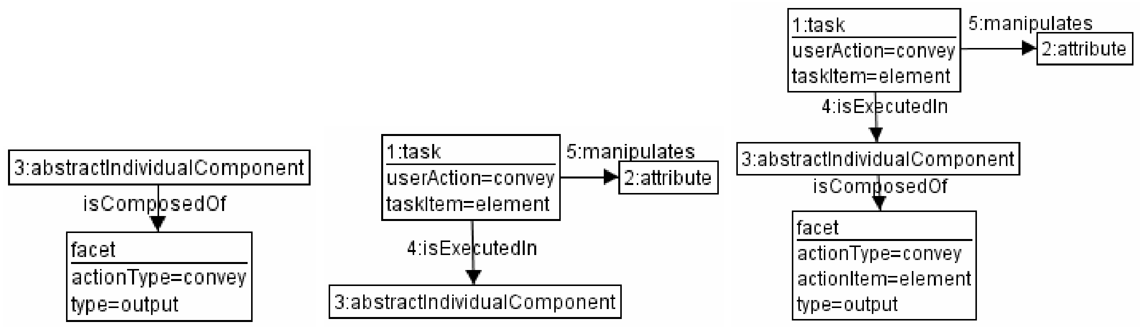


Figure 4-29. Create an output facet that conveys an element

After generating the AICs in the previous sub-step, the current sub-step identifies their types by associating a suitable facet. This sub-step considers three design options:

- Navigation concretization
- Control concretization
- Concretization of navigation and control

Navigation concretization. If a start operation task is manipulating a method from the domain concept, the AIC will be endowed with a navigation facet of type start operation (Figure 4-30).

NAC LHS RHS

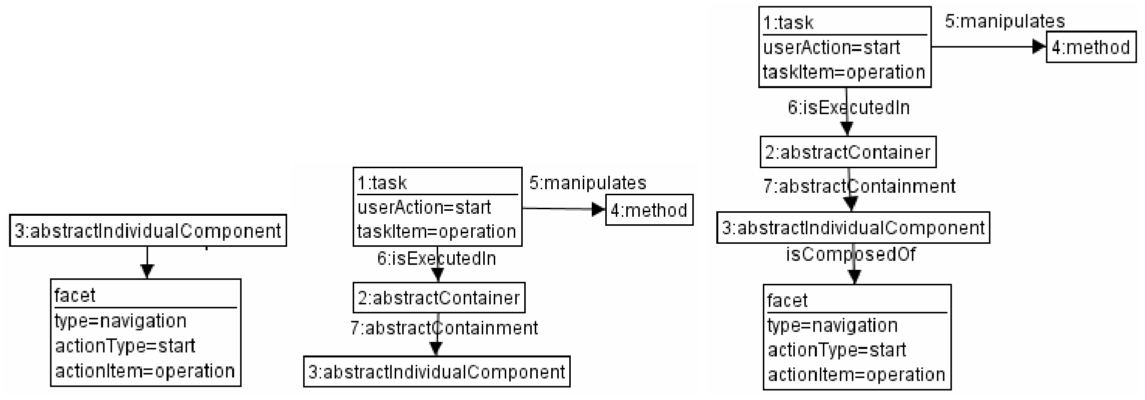


Figure 4-30. Create of navigation facet for AICs

Control concretization. A control facet will be generated by applying the rule described in Figure 4-31. In this case the start operation task is manipulating an attribute from the domain concepts.

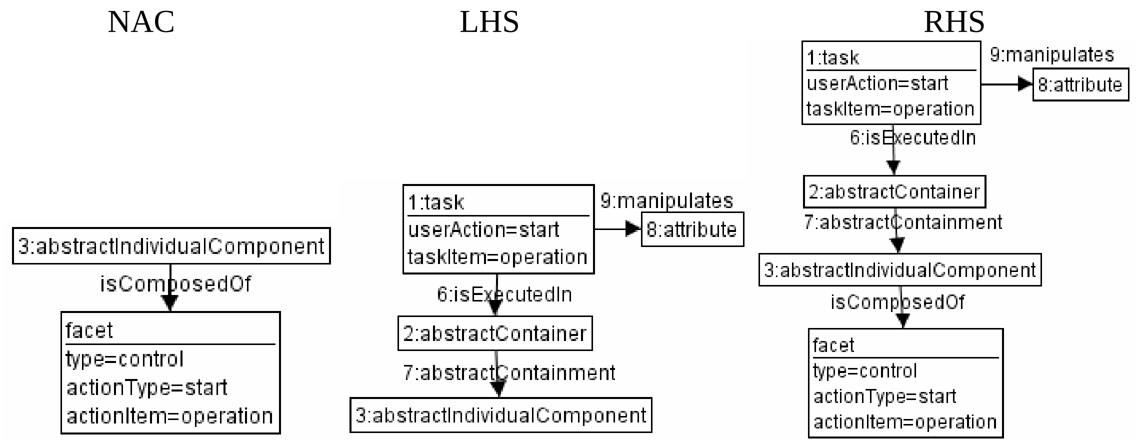


Figure 4-31. Create a control facet for AICs

Navigation and control concretization. The designer considers two possible options:

- Ensure the control and the navigation with *separated* objects (i.e., AICs can have either a navigation facet either a control one)
- Ensure the control and the navigation with the same object which *combines* simultaneously the two functions (i.e., AICs have two facets, one of type navigation and one of type control). For instance, applying the last two rules ensures a combined navigation and control by generating two facets, one of type navigation and one of type control for the same AIC.

4.3.2.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 of AIOs. For this purpose, the abstractAdjacency relationship is employed (see Section 3.1.3).

For each couple of sister tasks executed into AIOs, we define abstractAdjacency relationships between these AIOs. As AIOs can be of two types (i.e., ACs or AICs), there are four possible combination to take into account. For each combination a specific rule is considered. Moreover, in order to perform a complete arrangement, a rule should be defined for each type of temporal relationships between the tasks.

Figure 4-32 illustrates the rule that creates relationships of type "abstractAdjacency" between each couple of ACs mapped into sister tasks connected by a sequential (">>") temporal relationship.

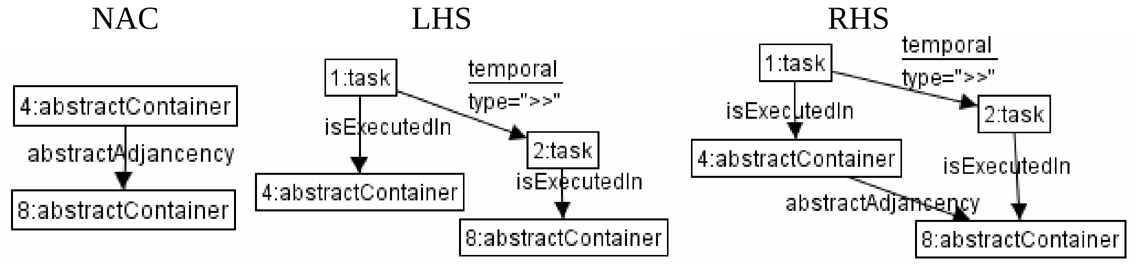


Figure 4-32. Creating abstract adjacency for <AC, AC> couple

4.3.2.d Rules for the definition of abstract dialog control

This sub-step is transposing the temporal relationships defined between the 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. Dialog control consists of controlling certain states of the user interface in order to enforce temporal constraints imposed between elements of the interface.

In order to ensure the abstract dialog control we employ the auiDialogControl relationship defined in Section 3.1.3. For each couple of sister tasks executed into AIOs, we define an abstractDialogControl relationship between these AIOs 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 take into account. Figure 4-33 exemplifies a rule applied to generate an auiDialogControl relationship between two ACs.

NAC LHS RHS

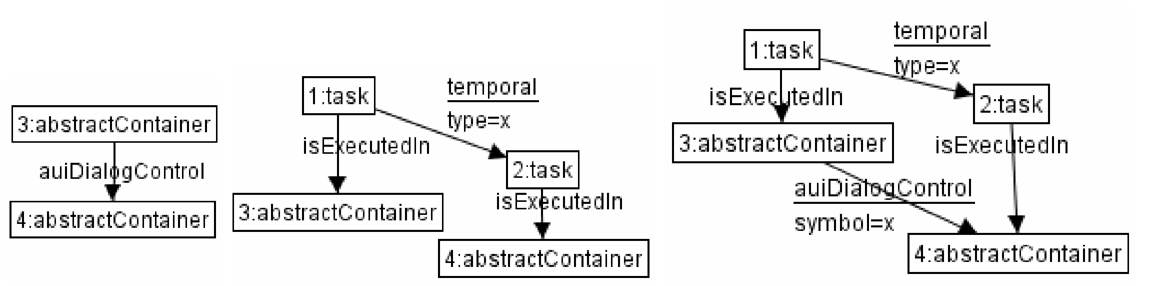


Figure 4-33. Transposing auiDialogControl relationship for <AC, AC> couple

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

This sub-step consists in refining the *manipulates* relationship defined between elements of the Task Model and elements of the Domain Model (see Section 3.1.7) into relationships between AICs from Abstract Model and elements of the Domain Model. The two refined relationship taken into consideration in the current dissertation are *updates* and *triggers* (see Section 3.1.7).

Figure 4-34 illustrates the rule used to synchronize the AICs with the attribute of an object from the Domain Model. The rule is applied for each task that manipulates an attribute. The synchronization is ensured by *updates* mapping relationship.

NAC LHS RHS

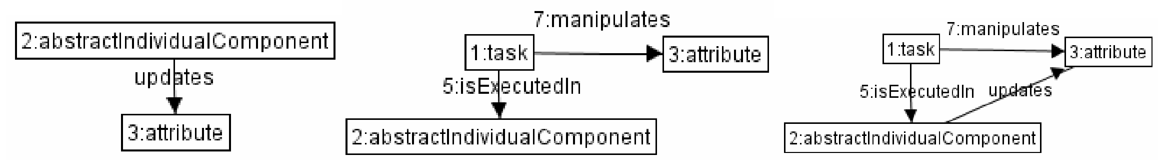


Figure 4-34. Deriving updates relationships for AICs

For each task that manipulates a method, the AIC that executes the task will trigger the method. The rule described in Figure 4-35 is generating a mapping relationship of type *triggers* between the AIC and the method.

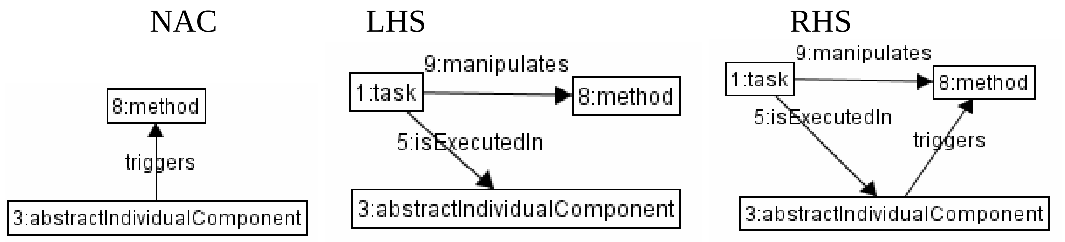


Figure 4-35. Deriving trigger relationships for AICs

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

The third transformation step involves a transformation system (see Section 4.1) that contains rules applied in order to realize the transition from the Abstract User Interface Model to the Concrete User Interface Model. It consists of a set of development sub-steps that are applied following a top-down logical order.

4.3.3.1 Selection of modality

The Concrete User Interface Model aims at defining a UI at a specification level that is dependent of the modality. Is now that the designer has to select which are the available modalities that will be employed in order to allow the interaction between the system and the user. The selection of the modalities takes into consideration a series of aspects connected to the context of use of the UI (see Section 3.1.6). Thus, the designer has to decide between three different cases of UIs (Figure 4-17):

- *Case 1: From AUI Model to Graphical CUI Model:* the only available modality is the graphical one
- *Case 2: From AUI Model to Vocal CUI Model:* only the vocal modality ensures the system-user interaction
- *Case3: From AUI Model to Multimodal CUI Model:* both, graphical and vocal modalities are employed in the system-user interaction. This dissertation do not considers other interaction modalities, such as gesture, haptic, tactile, etc.

4.3.3.2 Design option for the selected modality

For each type of Concrete User Interface identified in the previous Section, a specific set of transformation sub-steps ise applied. Moreover, we associate the design options defined in Section 4.2 to the different identified sub-steps.

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 classified in six development sub-steps (see Figure 4-18).

4.3.3.a Reification of AC into CC

This sub-step is dedicated to the reification of AC into GC. In Section 4.3.2.a, the identification of ACs was based on the **sub-task presentation** design option. As in the current step we generate a specification that is modality-dependent, we consider the values of the design options that can be found at the Concrete level of the Figure 4-10. Thus, in the current sub-step we exploit the different possible final representations of the sub-task presentation design option for graphical UIs.

Sub-task presentation. If the designer's choice is for a separated presentation of the sub-tasks then rule illustrated in Figure 4-36 can be applied. The rule creates for each AC corresponding to a sub-task a GC of type window that will embed a GC of type box.

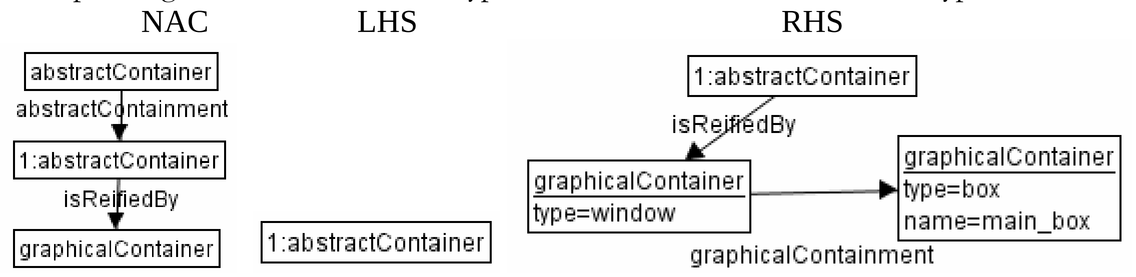


Figure 4-36. Creation of a window and the corresponding main box for each top-level AC

In the case of combined sub-task presentation, the designer has three different types possible combinations: (1) one by one, (2) many at once or (3) all in one. For each combination a set of final graphical presentation of the sub-task has been identified (see Section 4.2.1). If the designer chooses, for instance, a combined all in one sub-task presentation under the form of group boxes, two rules have to be applied in order to achieve the objective. First, the rule illustrated in Figure 4-36 will be applied, reifying the top most AC into a GC of type window that embeds a GC of type box. Further, each AC contained into the top-most AC, is reified into a GC of type groupBox (Figure 4-37).

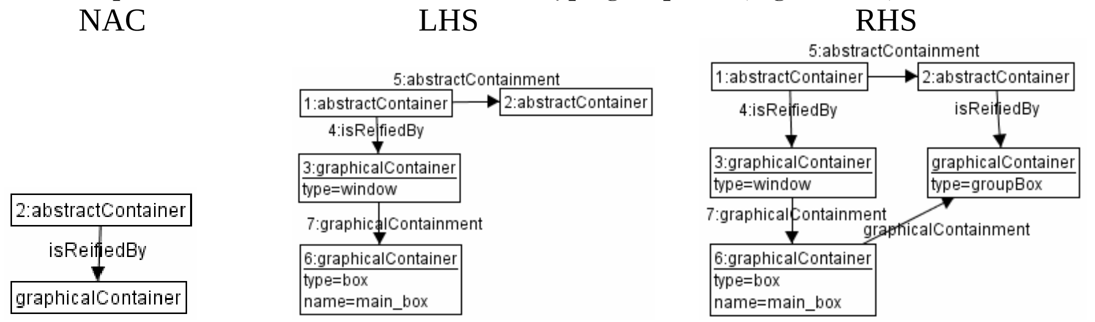


Figure 4-37. Create a group box for each AC embedded into the top-most AC

4.3.3.b Selection of CICs

This sub-step supposes the identification of graphical concrete elements that are the most suitable to support the functionalities of AICs ensured by their facets identified in Section 4.3.2.b.

Table 4-2 provides mappings between AICs defined by their facets and GICs in which they can be reified. The table does not cover all possible combinations of actions and items that correspond to a facet, it takes into account only those that are used in the current dissertation. The left column identifies the combinations of *actionType* and *actionItem* attributes of AIC facets, while the right column shows the corresponding GIC type.

AIC facet **facet type + (actionType + actionItem)**	**GIC type**
Control + (start + operation)	button
Navigation + (start + operation)	button
Input + (select + element)	radioButton OR checkBox OR comboBox

	item OR listBox item
Input + (create + element)	inputText
Output + (convey + element)	outputText

Table 4-2. Mappings between facet types and GIC types

In order to exemplify one of the mappings described in the above table, Figure 4-38 presents the rule applied to generate a GC of type **outputText**, each time when an output facet of type create element is encountered.

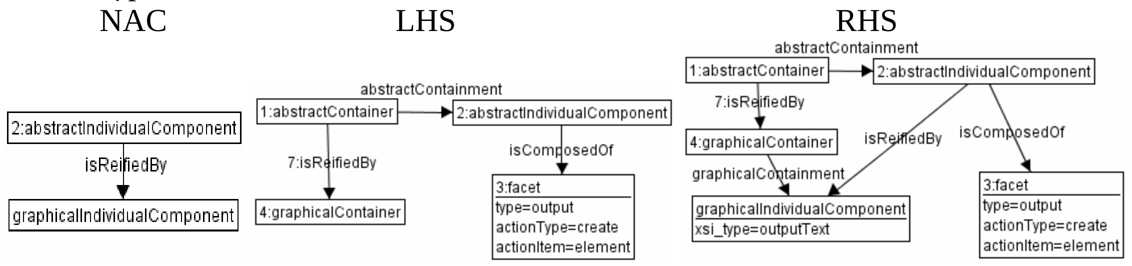


Figure 4-38. Create an outputText when an output facet of type create element is found

4.3.3.c Arrangement of CICs

This sub-step is applied in order to provide the concrete layout information of the UI. It consists in a transposition of the abstractAdjacency relationship defined between each couple of AIOs (see Section 4.3.2.c) into a graphicalAdjacency relationship between graphicalCIOs that reify them. As AIOs can be of two types (i.e., ACs or AICs), there are four possible combination to take into account. For each combination a specific rule is considered. Figure 4-39 illustrates the rule that transposes the abstractAdjacency relationship between two ACs into a graphicalAdjacency relationship between the GCs that reify these ACs.

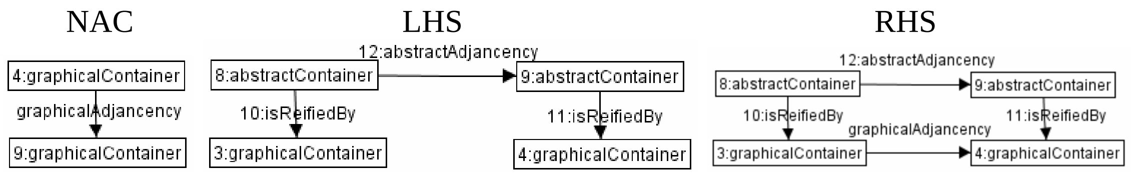


Figure 4-39. Generation of Graphical Adjacency relationships for <GC, GC> couples

4.3.3.d Navigation definition

This sub-step aims at specifying the navigation structure among the different GCs populating a UI. In Section 4.3.2.a the generation of AIC that will ensure the navigation between containers was based on the sub-task navigation. In Section 4.3.3.b the AICs were reified in their corresponding GIC.

Sub-task navigation. In the current sub-step the designer can define the navigation type between GCs by endowing the GICs that ensure the navigation with graphical transition features. There are two possible values of navigation: sequential and asynchronous. For exemplification we consider three sub-tasks executed each one in a corresponding groupBox. All group boxes are combined into the same window (Figure 4-40). The navigation between the group boxes is ensured by the Ok and Cancel buttons. If the designer's choice is for a sequential navigation, the following scenario should be ensured: once the user has fulfilled the information required in the group box corresponding to sub-task 1, only the sub-task 2 can be activated (see navigation a). From the group box associated to sub-task 2, there are two possibilities: returning to the sub-task 1 group box (see navigation c) or continue with the sub-task 3 (see navigation b). From sub-task 3 group box the user can activate only the sub-task 2 group box (see navigation d).

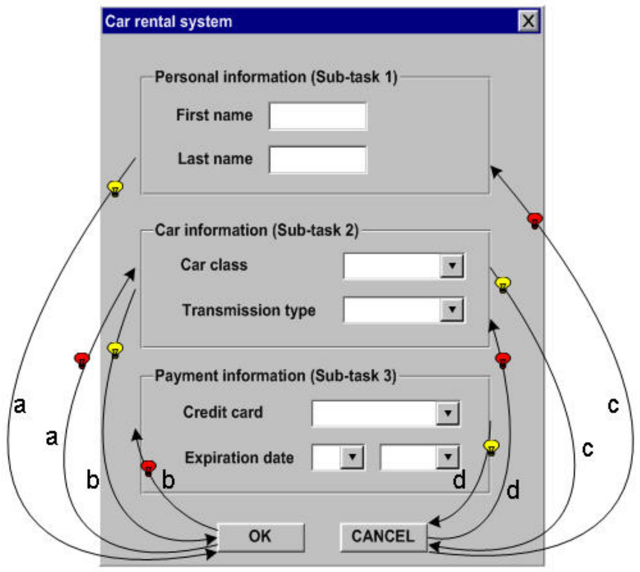


Figure 4-40. Sub-tasks presented into combined group boxes

In order to ensure the sequential navigation described above, rule illustrated in Figure 4-41 is applied. The rule is generating graphicalTransition relationships that endow the *Ok* and *Cancel* buttons with *activation* and *deactivation* power over the adjacent and current GC, respectively.

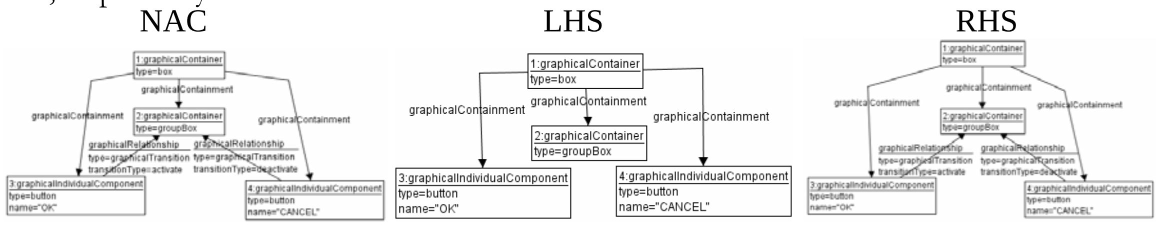
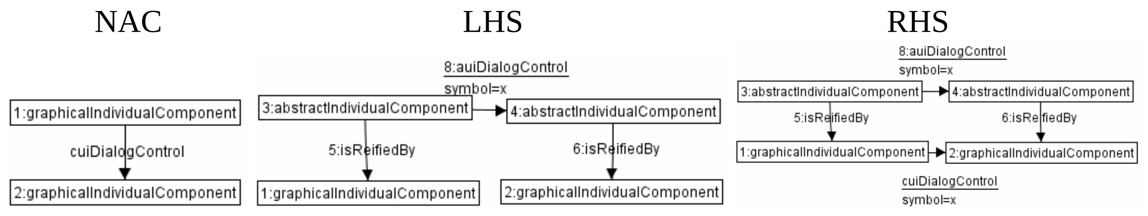


Figure 4-41. Endow the navigation buttons with graphicalTransition features

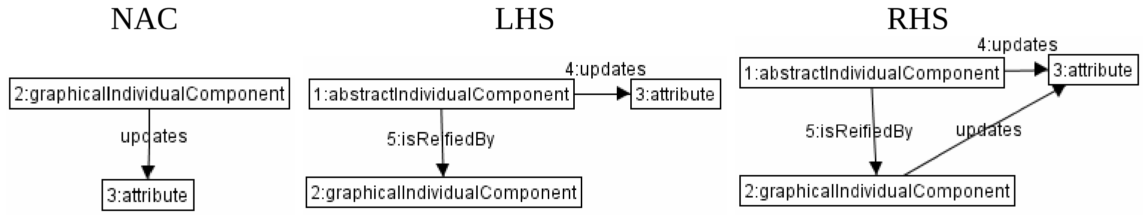
4.3.3.e Concrete dialog control definition

This sub-step realizes a transposition of auiDialogControl relationships defined between each couple AIOs (see Section 4.3.2.d) into cuiDialogControl relationships between graphicalCIOs that reify them. As AIOs are of two types (i.e., ACs and AICs), four rules describing the four possible combinations are considered. Figure 4-42 illustrates the rule that transposes the auiDialogControl relationship between two AICs into a cuiDialogControl relationship between the GICs that reify these AICs.


NAC
LHS
RHS
1:graphicalIndividualComponent
cuiDialogControl
2:graphicalIndividualComponent
8:auiDialogControl
symbol=x
3:abstractIndividualComponent
4:abstractIndividualComponent
5:isReifiedBy
6:isReifiedBy
1:graphicalIndividualComponent
2:graphicalIndividualComponent
cuiDialogControl
symbol=x
Figure 4-42. Generation of Concrete Dialog Control relationships for <GIC, GIC> couples

4.3.3.f Derivation of CUI to domain relationships

This step aims at transposing the relationships defined in Section 4.3.2.e to the concrete level. Thus, relationships between GICs and domain objects will be defined. Figure 4-43 illustrates the rule used to map GICs with the corresponding attribute of an object from the Domain Model. The **updates** relationship is transposed from the AIC that is reified by the GIC.


NAC
LHS
RHS
2:graphicalIndividualComponent
updates
3:attribute
4:updates
1:abstractIndividualComponent
3:attribute
5:isReifiedBy
2:graphicalIndividualComponent
updates
Figure 4-43. Transposition of update relationship

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 classified in six development sub-steps (see Figure 4-18).

4.3.4.a Reification of AC into CC

This sub-step is dedicated to the reification of AC into VC. By analogy with Section 4.3.3.a, the current sub-step exploits the different possible final representations of the sub-task presentation design option for vocal UIs.

Sub-task presentation. If the designer's choice is for a separated presentation of the sub-tasks, then the rule illustrated in Figure 4-44 can be applied. The rule creates for each AC corresponding to a sub-task, a VC of type vocalGroup.

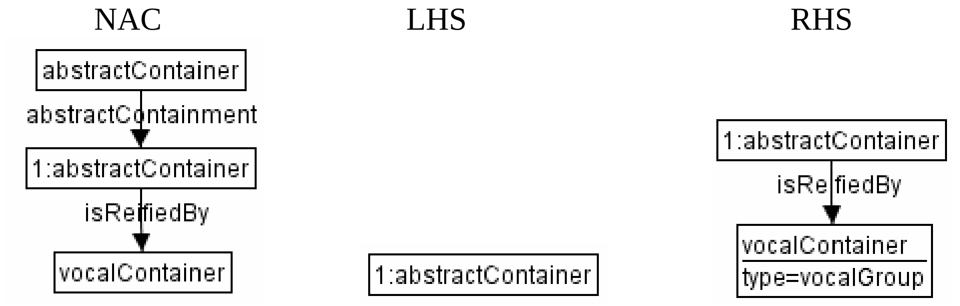


Figure 4-44. Creation of a VC for each top-level AC

In the case of combined sub-task presentation, if the designer considers the value *all in one,* he has to specify further the value of his choice. If for graphical UIs we have identified a series of possible values (see Section 4.2.1), for vocal UIs the identification of the values is difficult to achieve due to the nature of vocal applications. For instance, an equivalent of the bulleted list value in vocal UI could be the following sequence: *"Choose one or more of these options: (beep) + Personal information + (3 seconds pause) + (beep) + Car information + 3 seconds pause + (beep) + Payment information"*, while the equivalent of the numbered list could be the sequence: *"Choose one or more of these options: (dong) + Personal information + (3 seconds pause) + (dong) + Car information + 3 seconds pause + (dong) + Payment information"*.

The fulfillment of vocal sub-tasks presented in an *all in one combined* mode is also difficult to achieve due to the sequential nature of vocal applications. An *all in one* value of the sub-task presentation for a vocal UI supposes the simultaneously availability of two or more vocal sub-tasks. For instance, if we refer to the example provided in Figure 4-39, in a vocal UI the user should be able to fulfill the personal information and the car information in the same time by saying: *"My name is Juan and the car transmission type is manual"*. From the theoretical point of view the realization of the sub-tasks is feasible, but from the usability point of view it is hard to achieve due to the limitations of the FUI.

However, we provide in the following the two rules that ensure a combined all in one presentation of the vocal sub-tasks. First, the rule illustrated in Figure 4-45 will be applied, reifying the top most AC in VC of type vocalGroup. Further, each AC contained in the top-most AC is reified into a VC of type vocalMenu (Figure 4-46).

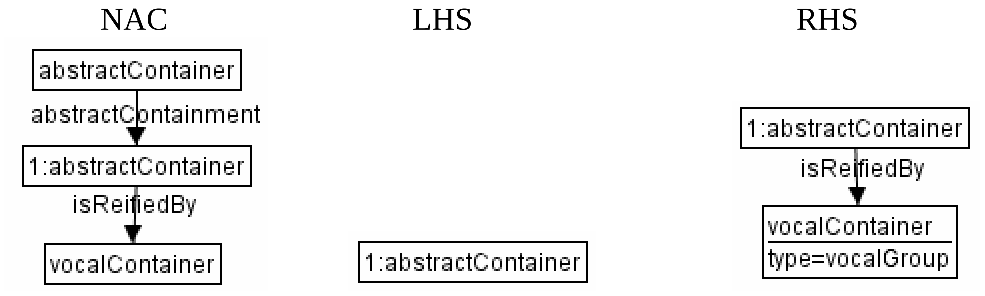


Figure 4-45. Creation of the vocal group of the UI

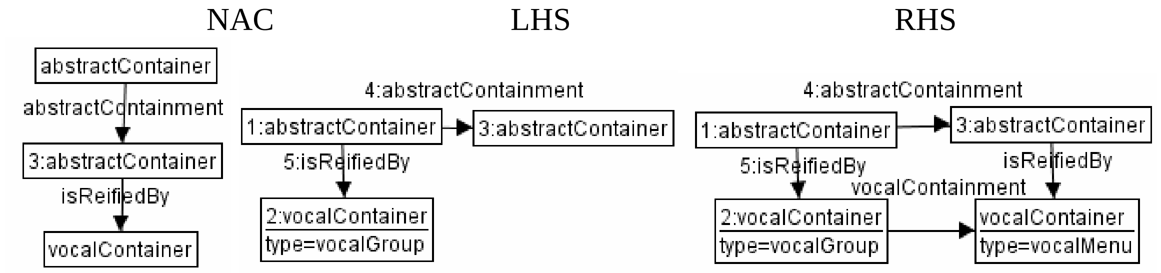


Figure 4-46. Creation of vocalMenus for each AC embedded into the top-most AC

4.3.4.b Selection of CICs

By analogy with Section 4.3.2.b, we provide in Table 4-3 the mappings between AICs defined by their facets and the VICs in which they can be reified. The left column identifies the combinations of actionType and actionItem attributes of AIC facets, while the right column shows the corresponding VIC type. The rules applied to generate a VICs are obtained by a simple replacement of GICs from the rules described in Section 4.3.2.b with the corresponding VIC identified in Table 4-3.

AIC facet **facet type + (actionType + actionItem)**	**VIC type**
Control + (start + operation)	vocalInput
Navigation + (start + operation)	vocalNavigation
Input + (select + element)	vocalMenuItem + vocalInput
Input + (create + element)	vocalPrompt + vocalInput
Output + (convey + element)	vocalPrompt

Table 4-3. Mappings between facet types and VIC types

For the rest of the sub-steps: **4.3.4.c Arrangement of CICs, 4.3.4.d Navigation definition, 4.3.4.e Concrete dialog control definition, 4.3.4.f Derivation of CUI to domain relationships,** the design of the rules is realized by analogy with the correspondent sub-step of the graphical rules presented in Case 1. Thus, the GCs and GICs will be replaced by VC and VICs, respectively.

Case3: From AUI Model to Multimodal CUI Model

The current case aims at deriving Multimodal Concrete UIs from Abstract UI specification by applying a set of transformational rules classified in seven development steps (see Figure 4-18). The rules applied in Case 1 for graphical UIs are joined with the rules applied in Case 2 for vocal UIs, resulting new rules that will ensure the generation of multimodal UIs. This will show the modularity and the extensibility of the transformation rules [Stan05] by describing how vocal components are added to the already existing graphical components.

4.3.5.a Reification of AC into CC

As described in the homologous sub-steps for graphical and vocal UIs (see Section 4.3.3.a and 4.3.4.a, respectively), the rules that ensure the current sub-step takes into consideration the different possible final representations of the sub-task presentation design option.

Sub-task presentation. If the designer's choice is for a separated sub-task presentation of the UI, the rule that will ensure this option is obtained by combing the rule for graphical UIs illustrated in Figure 4-36 and the rule described for vocal UIs in Figure 4-44. The resultant rule is presented in Figure 4-47.

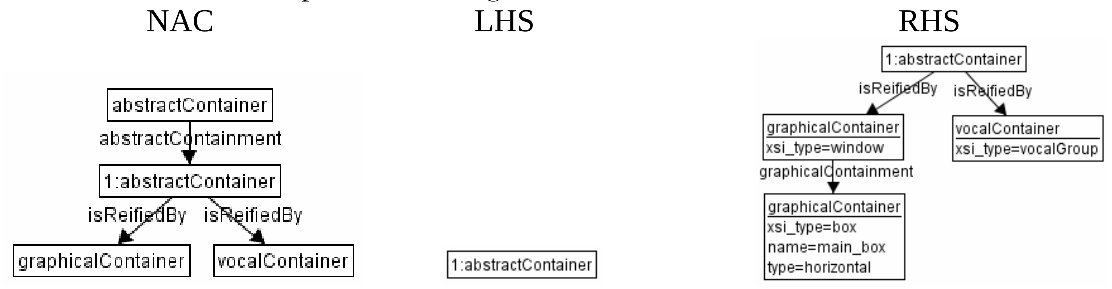


Figure 4-47. Creation of a GC and a VC for each top-level AC

For combined all in one sub-task presentation under the form of group boxes, the designer applies two rules. First, the rule from Figure 4-47 ensures the reification of each top-level AC into a GC of type window that embeds a GC of type box and into a VC of type vocalGroup. Further, the rule for graphical UIs illustrated in Figure 4-37 is combined with the rule for vocal UIs from Figure 4-46. The resultant rule is presented in Figure 4-48.

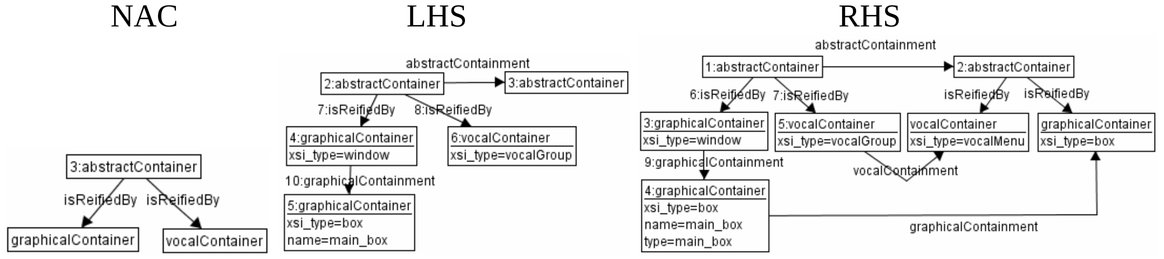


Figure 4-48. Creation of graphical and vocal containers embedded into the top most containers

4.3.5.b Selection of CICs

In order to identify the multimodal CICs that are the most suitable to support the functionalities of the AICs ensured by their facets, Table 4-4 provides a series of mappings used in this dissertation. The table results from a combination of values presented in Tables 4-2 and 4-3. Please notice that the CICs provided in brackets ([ ]) are optional.

AIC facet **facet type + (actionType + actionItem)**	**GIC and VIC types**
Control + (start + operation)	button + vocalInput + [vocalFeedback]
Navigation + (start + operation)	button + vocalNavigation + [vocalFeedback]
Input + (select + element)	radioButton OR checkBox OR comboBox OR listBox item + vocalMenuItem + vocalInput+ [vocalFeedback]
Input + (create + element)	inputText + vocalPrompt + vocalInput + [vocalFeedback]
Output + (convey + element)	outputText + vocalPrompt

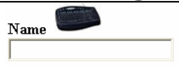
Table 4-3. Mappings between facet types and GIC and VIC types

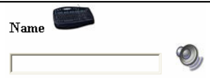
In the current sub-step the designer takes into consideration the four multimodal design options identified in Section 4.2.3:

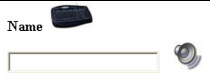
- Prompting
- Grouping/Distinction of Items
- Immediate feedback
- Guidance

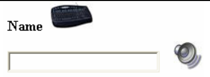
Table 4-4 summarizes all possible renderings for combinations of *Grouping for input* and *Feedback* design options for a text input in the context of Web UIs. As the current dissertation addresses only graphical and vocal interactions, only these two modalities are taken into account, but the proposition can be extended to other interaction modalities. Depending of each combination, a specific guidance for input and feedback will be associated.

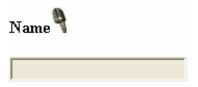
Design options	**Rendering**

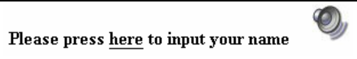
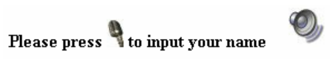
Input = graphical (assignment)
Feedback = graphical (assignment)
	

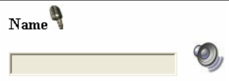
Input = graphical (assignment)
Feedback = multimodal (equivalence)
	

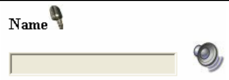
Input = graphical (assignment)
Feedback = multimodal (complementarity)
	

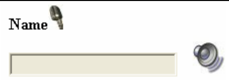
Input = graphical (assignment)
Feedback = multimodal (redundancy)
	

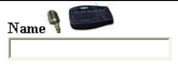
Input= vocal (assignment)
Feedback= graphical (assignment)
	

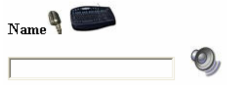
Input= vocal (assignment)
Feedback= vocal (assignment)
(two different ways of displaying the prompt)
	 Please press here to input your name
	vs.
	 Please press to input your name

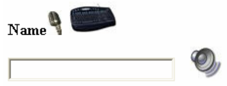
Input= vocal (assignment)
Feedback= multimodal (equivalence)
	

Input= vocal (assignment)
Feedback= multimodal (complementarity)
	

Input= vocal (assignment)
Feedback= multimodal (redundancy)
	

Input= multimodal (equivalence)
Feedback= graphical (assignment)
	

Input= multimodal (equivalence)
Feedback=multimodal (equivalence)
	

Input= multimodal (equivalence)
Feedback=multimodal (complementarity)
	

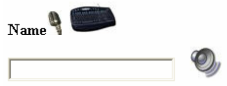
Input= multimodal (equivalence)
Feedback=multimodal (redundancy)
	

Table 4-4. Combinations of input and feedback design options values for a text input

We exemplify the design options considered in the current sub-step with a possible design decision (Figure 4-49) for a multimodal text input where the user has to provide her name [Stan06]. The value of the prompt design option is multimodal as the system indicates in a redundant way the task to fulfill by employing two modalities: graphical modality (the label *Name*) and vocal modality used by the systems to invite the user to input his name (1). The guidance for input is of type iconic and is composed of two elements (the microphone icon and the keyboard icon) indicating the available interaction modalities. User's input is of type multimodal as it can be provided in an equivalent manner by employing either the graphical modality (the user is typing his name in the text entry), either the vocal modality (the user is uttering his name using the microphone (2)). The guidance for feedback is of type iconic and is ensured by the loudspeaker icon, indicating the vocal feedback. The feedback of the system to the user's input is of type multimodal as it is expressed by means off two redundant modalities: graphical (the result of users' typing) and vocal (the system is uttering the result of the input recognition (3)).

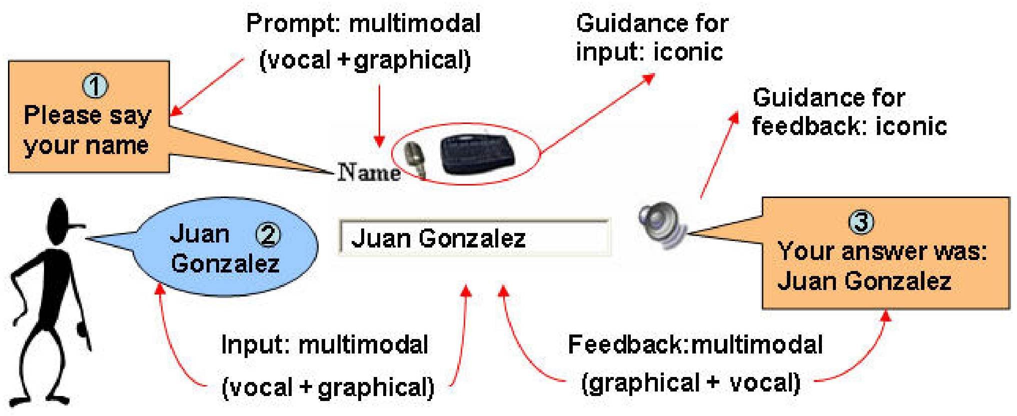

Prompt: multimodal (vocal + graphical)
Guidance for input: iconic
Guidance for feedback: iconic
① Please say your name
Name
Juan ② Gonzalez
Juan Gonzalez
③ Your answer was: Juan Gonzalez
Input: multimodal (vocal + graphical)
Feedback:multimodal (graphical + vocal)

Figure 4-49. A possible design decision for a multimodal text input

Table 4-5 presents the design options and their CICs concretization for the multimodal text input described above.

Design option	Value	CIC
Prompting	Multimodal (redundancy)	outputText + vocalPrompt
Grouping for input	Multimodal (equivalence)	inputText + vocalInput
Immediate feedback	Multimodal (redundancy)	inputText + vocalFeedback
Guidance for input	Iconic (assignment)	imageComponents (keyboard icon + microphone icon)
Guidance for feedback	Iconic (assignment)	imageComponent (speakerIcon)

Table 4-5. Design option values for multimodal textInput widget (graphical and vocal equivalence for input)

Rule illustrated in Figure 4-50 generates the multimodal text input described above by creating a GC of type box that embeds two GICs: an inputText and an outputText (the label associated to the inputText). The feedback is ensured by a VIC of type vocalFeedback. The guidance for input is ensured by two GICs of type imageComponent represented with keyboard and microphone icons, while the guidance for output is represented with a speaker icon.

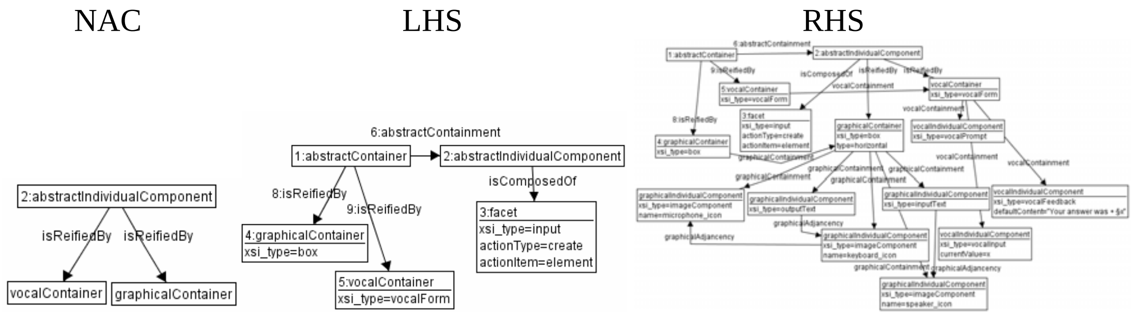

NAC
LHS
RHS
2:abstractIndividualComponent
isReifiedBy
isReifiedBy
vocalContainer
graphicalContainer
6:abstractContainment
1:abstractContainer
2:abstractIndividualComponent
8:isReifiedBy
9:isReifiedBy
isComposedOf
4:graphicalContainer xsi_type=box
5:vocalContainer xsi_type=vocalForm
3:facet xsi_type=input actionType=create actionItem=element
graphicalContainer xsi_type=box type=horizontal
vocalContainer xsi_type=vocalForm
vocalContainment
graphicalContainment
graphicalAdjancency
graphicalIndividualComponent xsi_type=imageComponent name=microphone_icon
graphicalIndividualComponent xsi_type=outputText
graphicalIndividualComponent xsi_type=inputText
graphicalIndividualComponent xsi_type=imageComponent name=keyboard_icon
graphicalIndividualComponent xsi_type=imageComponent name=speaker_icon
vocalIndividualComponent xsi_type=vocalPrompt
vocalIndividualComponent xsi_type=vocalFeedback defaultContent="Your answer was + $x"
vocalIndividualComponent xsi_type=vocalInput currentValue=x

Figure 4-50. Generation of a multimodal inputText (graphical and vocal equivalence for input)

If the designers' decision is for a multimodal text input that differentiate from the above described example by (Table 4-6):

- The grouping for input changes from multimodal equivalence into vocal assignment

- The immediate feedback changes from multimodal redundancy into graphical assignment
- There is no more guidance for input,

then the rule illustrated in Figure 4-51 should be applied.

Design option	Value	CIC
Prompting	Multimodal (redundancy)	outputText + vocalPrompt
Grouping for input	Vocal (assignment)	vocalInput
Immediate feedback	Graphical (assignment)	inputText
Guidance for input	Iconic (assignment)	imageComponents (microphone icon)

Table 4-5. Design option values for multimodal textInput widget (vocal modality assigned for input)

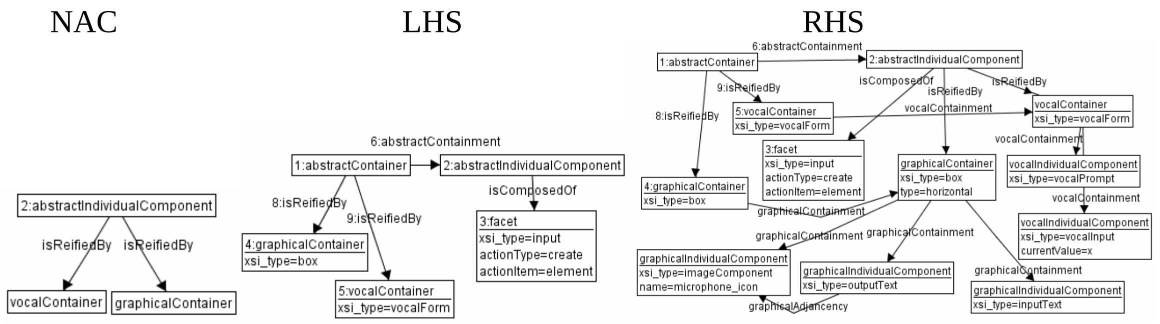


Figure 4-51. Generation of a multimodal inputText (vocal modality assigned for input)

4.3.5.c Synchronization of CICs

Comparing with the previous two cases (i.e., generation of graphical and vocal CUI Models from the AUI Model), the current case dedicated to the generation of multimodal CUI Models is introducing a new sub-step (Requirement 15. Methodological extendibility). This sub-step aims at ensuring the coordination of vocalCIOs and the graphicalCIOs by generating a synchronization relationship between them (see Section 3.2.4).

Hereafter we exemplify the synchronization relationship with two examples of rules that allow its generation. If the designer wants to allow users to interact with a combobox widget by employing the vocal modality, then he must ensure the synchronization between the vocalInput that will gather the input from the user and the combobox. Thus, the rule illustrated in Figure 4-52 generates a synchronization relationship between the VIC of type *vocalInput* and the GIC of type *comboBox*. The synchronization is defined between the currentValue *x* of the VIC and the currentValue *y* of the GIC.

Th second example corresponds to the designer's decision of allowing users to interact vocally with a text field. Thus, synchronization between the vocal input gathered from the used and the inputText must be ensured. In order to reach this objective rule illustrated in Figure 4-53 can be applied. It creates the synchronization relationship between the VIC of type *vocalInput* and GIC of type *inputText*. The synchronization is defined between the currentValue *x* of the VIC and the currentValue *y* of the GIC.

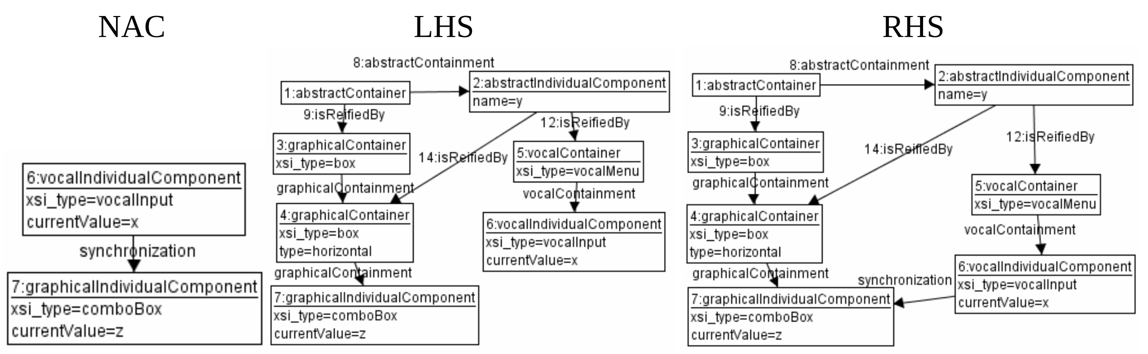

NAC
LHS
RHS
6:vocalIndividualComponent xsi_type=vocalInput currentValue=x
synchronization
7:graphicalIndividualComponent xsi_type=comboBox currentValue=z
8:abstractContainment
1:abstractContainer
2:abstractIndividualComponent name=y
9:isReifiedBy
12:isReifiedBy
14:isReifiedBy
3:graphicalContainer xsi_type=box
5:vocalContainer xsi_type=vocalMenu
graphicalContainment
vocalContainment
4:graphicalContainer xsi_type=box type=horizontal
6:vocalIndividualComponent xsi_type=vocalInput currentValue=x
graphicalContainment
7:graphicalIndividualComponent xsi_type=comboBox currentValue=z
synchronization

Figure 4-52. Synchronization between a vocalInput and a comboBox

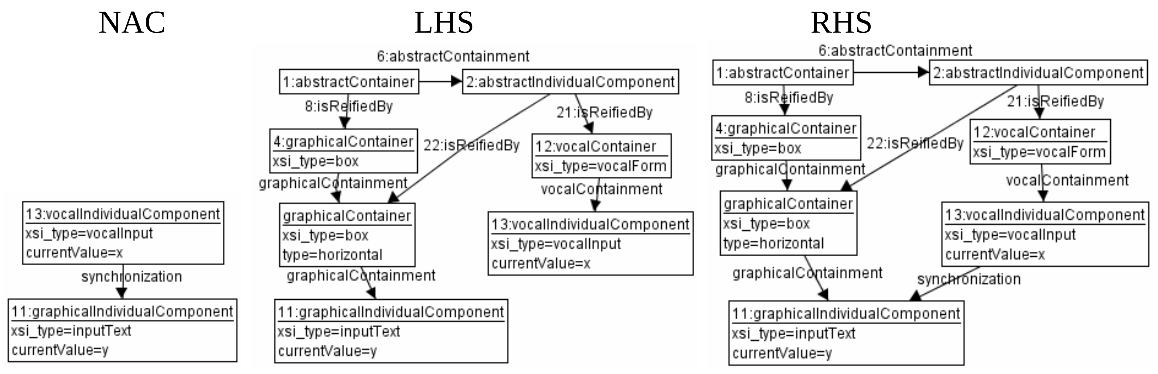

NAC
LHS
RHS
13:vocalIndividualComponent xsi_type=vocalInput currentValue=x
synchronization
11:graphicalIndividualComponent xsi_type=inputText currentValue=y
6:abstractContainment
1:abstractContainer
2:abstractIndividualComponent
8:isReifiedBy
21:isReifiedBy
22:isReifiedBy
4:graphicalContainer xsi_type=box
12:vocalContainer xsi_type=vocalForm
graphicalContainment
vocalContainment
graphicalContainer xsi_type=box type=horizontal
13:vocalIndividualComponent xsi_type=vocalInput currentValue=x
graphicalContainment
11:graphicalIndividualComponent xsi_type=inputText currentValue=y
synchronization

Figure 4-53. Synchronization between a vocalInput and an inputText

4.3.5.d Arrangement of CICs

The current sub-step supposes the arrangement of graphicalCIOs and vocalCIOs. The objective is achieved by applying the rules designed according to Section 4.3.3.c for graphicalCIOs and the rules illustrated in Section 4.3.4.c for vocalCIOs.

4.3.5.e Navigation definition

The current sub-step is achieved by applying the rules described in Section 4.3.3.d for graphical components of the multimodal UI and the rules designed according to Section 4.3.4.d for vocal components.

4.3.5.f Concrete dialog control definition

This sub-step is achieved by applying the rules described in Section 4.3.3.e for graphical components of the multimodal UI and the rules designed according to Section 4.3.4.e for vocal components.

4.3.5.g Derivation of CUI to domain relationships

This sub-step is achieved by applying the rules described in Section 4.3.3.f for graphical components of the multimodal UI and the rules designed according to Section 4.3.4.f for vocal components.

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

This step aims at generating the source code of the FUI from the CUI. For each type of CUI considered in the previous step (i.e., graphical, vocal and multimodal), a corresponding source code will be automatically generated. Thus, for graphical UIs we generate XHTML code, VoiceXML code is considered for vocal UIs, while multimodal UIs will be supported be XHTML+Voice language. Further, we interpret the generated code within a browser in order to obtain the corresponding FUI. We have used the following browsers:

- For graphical FUI: any ordinary web browser (e.g., Internet Explorer, Mozilla)
- Vocal FUIs: interpreted with IBM VoiceXML browser
- Multimodal FUIs: interpreted within NetFront multimodal browser or Opera browser. With respect to the CARE properties we consider only *Assignment*, *Equivalence* and *Redundancy*. The *input Complementarity* requires the system to perform *data fussion* whereas *output Complementarity* requires *data fission*. Neither *data fussion* nor *data fission* are currently supported by X+V browsers. Therefore we have no control over these aspects.

In order to support this step we present in the next Section several tools that have been developed to provide code generation from models.

4.4 Tool support

One of the main advantages of the design space introduced in Section 4.2 is given by the fact that each design option composing the space is independent of any existent method or tool, thus being useful for any developer of multimodal UIs. Under these circumstances, it would be useful to provide an explicit support of the introduced design options (Requirement 10. Machine processable).

In order to support the development of computer-aided design of MMWUIs, we consider MultiXML, an assembly of five software modules [Stan06]. By using MultiXML, we want to address a reduced set of concerns by limiting the amount of design options, thus making the design space more manageable and tractable [Hoov91]. Figure 4-54 illustrates the general development scenario of our transformational approach by identifying the five modules of MultiXML tool along with the steps in which they are employed. MultiXML offers the possibility of reusing the output provided by one module into another (Requirement 16. Support for tool interoperability).

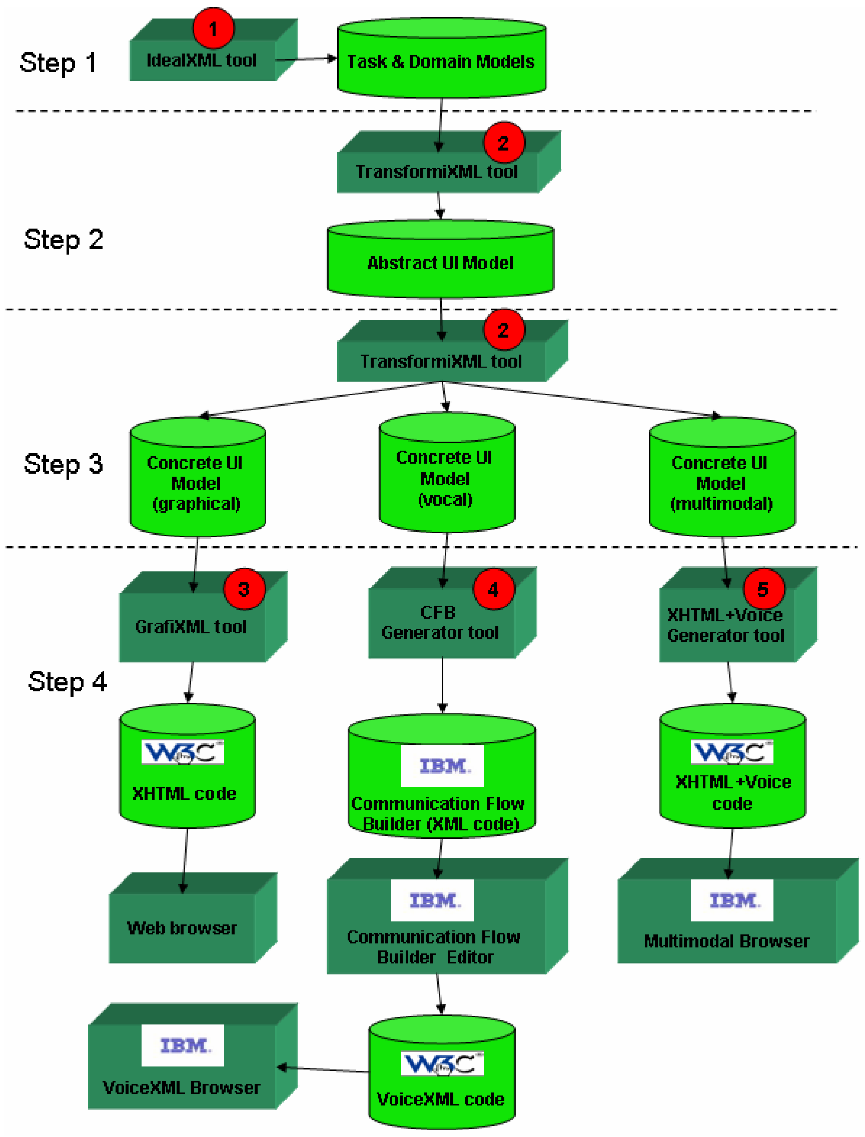


Figure 4-54. General development scenario – identification of MultiXML modules

4.4.1 IdealXML

IdealXML [Mont05] is a tool that is involved in the first step of the transformational approach and allows designers to graphical describe the Task Model, Domain Model 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 Task Model (Figure 4-55) takes the form of a CTT notation introduced by [Pate97]. The Domain Model (Figure 4-56) has the appearance of a class diagram, while the Mapping Model (Figure 4-57) is specified by associating graphically elements of the Task Model with elements of the Domain Model.

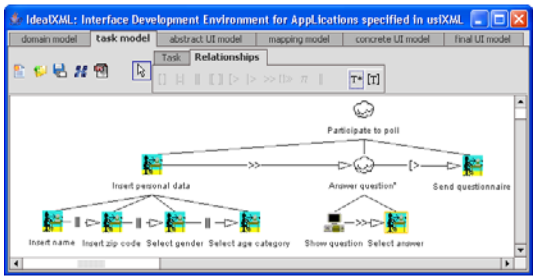


Figure 4-55. Task Model editor

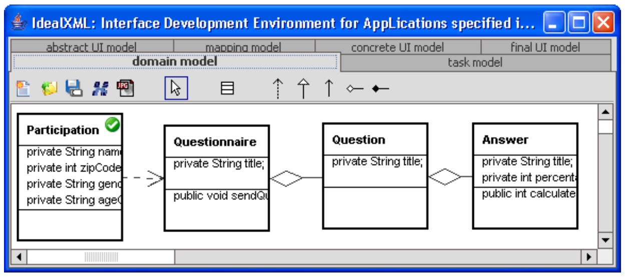
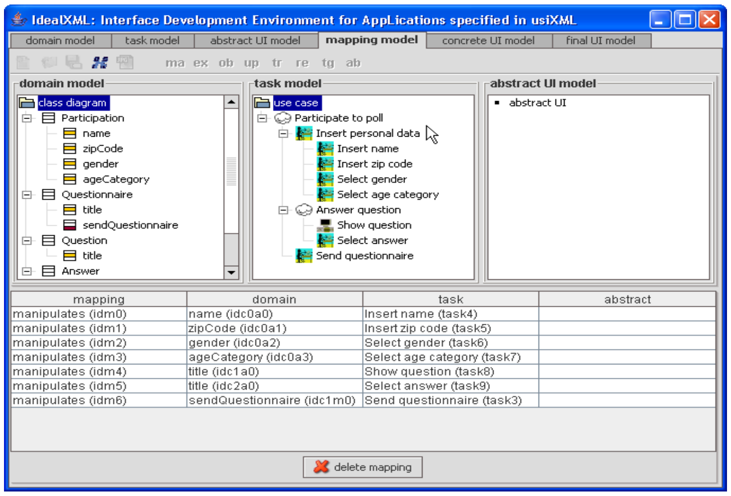


Figure 4-56. Domain Model editor



mapping	domain	task	abstract
manipulates (idm0)	name (idc0a0)	Insert name (task4)	
manipulates (idm1)	zipCode (idc0a1)	Insert zip code (task5)	
manipulates (idm2)	gender (idc0a2)	Select gender (task6)	
manipulates (idm3)	ageCategory (idc0a3)	Select age category (task7)	
manipulates (idm4)	title (idc1a0)	Show question (task8)	
manipulates (idm5)	title (idc2a0)	Select answer (task9)	
manipulates (idm6)	sendQuestionnaire (idc1m0)	Send questionnaire (task3)	

Figure 4-57. Mapping Model editor

4.4.2 TransformiXML

The transformation approach is sustained in steps 2 and 3 by TransformiXML, the core module of the MultiXML software that enables the definition and the application of transformation rules based on design options. The tool is sub-divided into two components:

- *TransformiXML API:* a Java Application Programming Interface that is employed to perform model-to-model transformations based on graph transformation rules. In order to ensure this function AGG API was selected due to our prior experience with the AGG tool [Ehri99]. In [Stan04] we developed and described an extension of the tool with an import and export function from and to a preliminary version of UsiXML models. Moreover, the tool was used as a transformation editor and interpreter in [Limb04b] in order to apply model-to-model transformations for the generation of graphical and vocal user interfaces. In the current dissertation we extend the application of transformations to multimodal user interfaces, too. The scenario of using AGG API to perform model-to-model transformations is described in [Limb04a] and consists of the following phases (Figure 4-58): the initial specification of a model along with a set of rules both expressed 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.

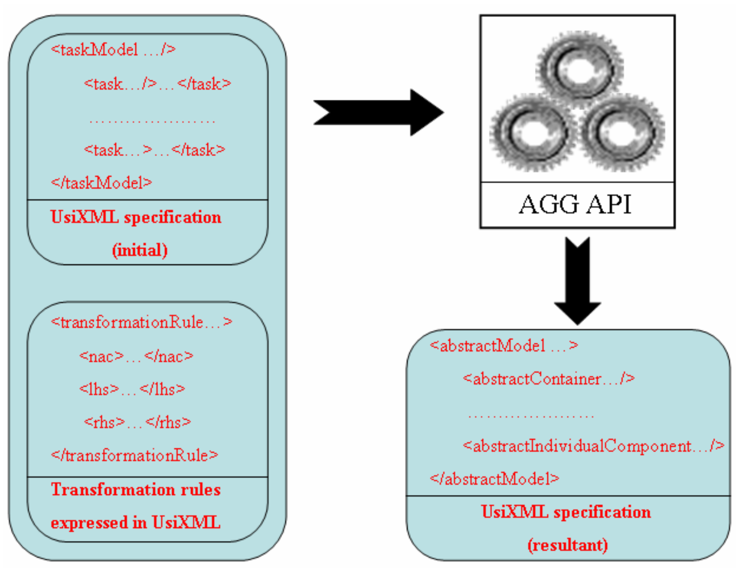


Figure 4-58. Model-to-model transformation based on AGG API

- *TransformiXML GUI:* is a graphical user interface that serves as a front-end application to the API. The basic flow of tasks with TransformiXML GUI (Figure 4-59) is the following: after choosing an input file containing models to transform, the user selects a development path by choosing a starting point and a destination point (e.g., the viewpoint to obtain at the end of the process). In the context of our dissertation the starting point is the Task and Domain Models and the destination is the AUI Model. All the steps and sub-steps of the chosen path can be visualized in a tree. By clicking on a sub-step in the tree, a set of transformation systems realizing the chosen sub-step are displayed. Each transformation system contains a set of rules that can be visualized in the Transformation rule explorer frame. The designer may also want to edit the rules either in GrafiXML editor ([Limb04b]) or in AGG tool. The transformations can be applied either step by step or from one shot. The result of the transformation is then explicitly saved in a UsiXML file.

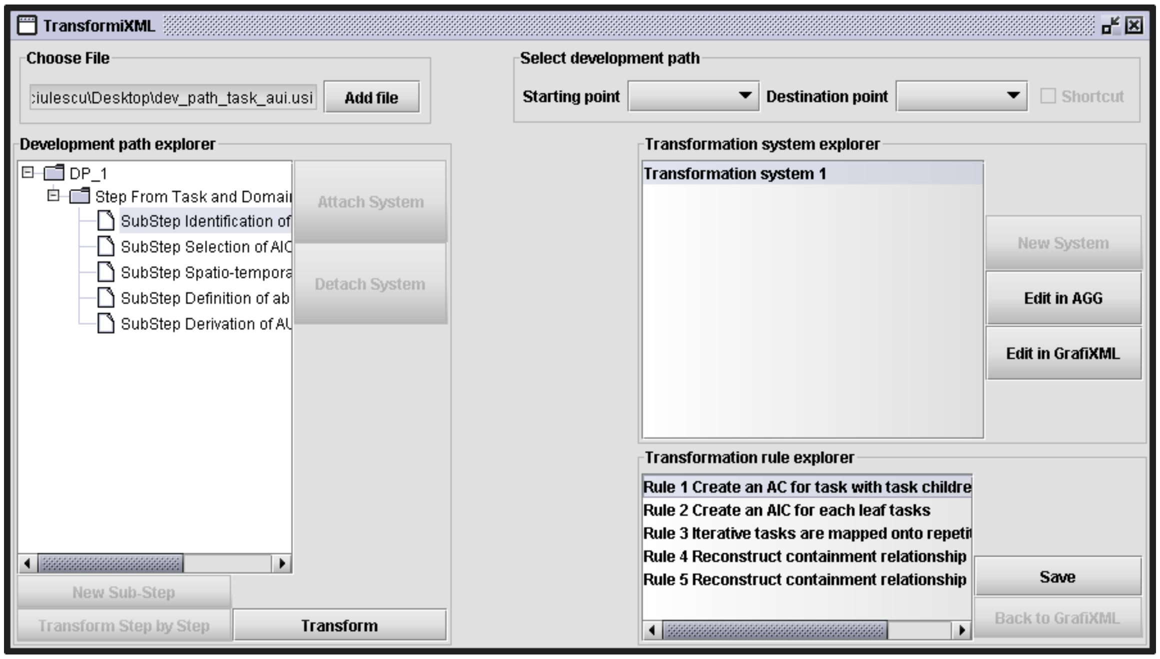


Figure 4-59. TransformiXML – graphical user interface

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 4-60 provides an example of a transformation rule applied manually over the initial *Task Model* (Figure 4-61) in order to generate the resultant AUI Model (Figure 4-62). The rule is creating AC in which each sub-task of the top-most task in a *Task Model* will be executed.

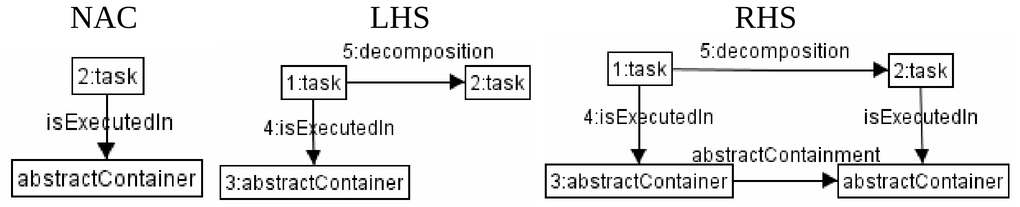
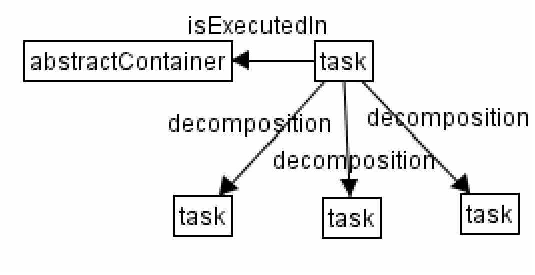


Figure 4-60. Generate abstract containers for each sub-task of the top-most task



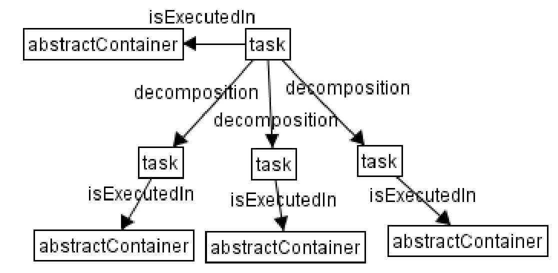


Figure 4-61. Initial Model

Figure 4-62. Resultant model

4.4.3 GrafiXML

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 4-63). 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 graphical UI by placing graphicalCIOs 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.

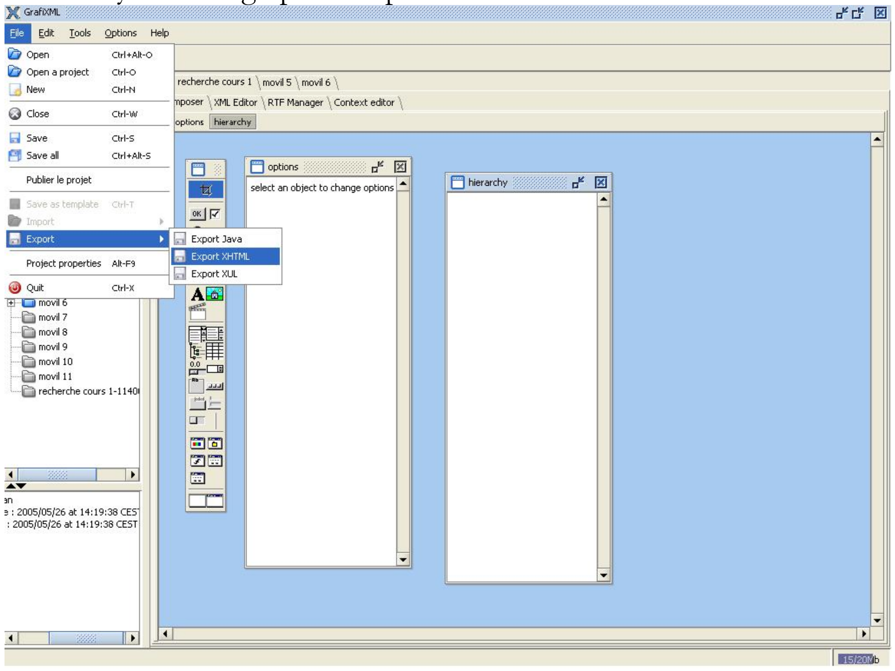


Figure 4-63. GrafiXML – export function

4.4.4 CFB (Communication Flow Builder) Generator

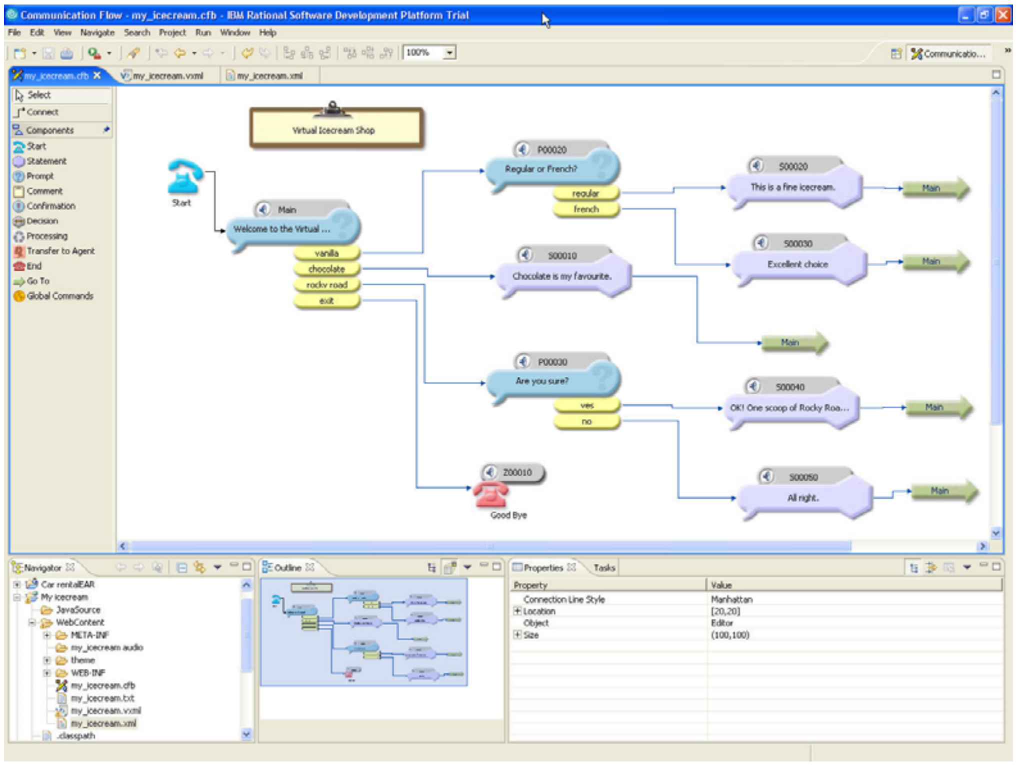
CFB Generator tool is the MultiXML module that is involved in step 4 of the transformational approach. It generates XML code corresponding to the Communication Flow Builder tool [IBM05] file format by applying XSL Transformations over the Concrete Vocal UI specification of UsiXML.

4.4.5 XHTML+Voice Generator

XHTML+Voice Generator tool is a MultiXML module software involved in step 4 of the transformational approach. It generates XHTML+Voice code by applying XSL Transformations over the multimodal specification of the Concrete UI Model.

4.4.6 Communication Flow Builder

For the vocal UIs we are not generating the VoiceXML code using our own tools, but we are employing IBM Communication Flow Builder (Figure 4-64), a graphical editor integrated in IBM Voice Toolkit that allows importing the results of the transformations applied by CFB Generator tool. Our decision of using a tool that is not one of the modules of the MultiXML software is sustained by the easy-to-use graphical interface of the editor that enables users to drag and drop graphical objects to create a communication flow. The editor also allows designers to generate VoiceXML code from the communication flow design and to test it in a VoiceXML browser.



Property	Value
Connection Line Style	Manhattan
Location	[20,20]
Object	Editor
Size	(100,100)

Figure 4-64. Communication Flow Builder editor

4.5 Conclusions

The current chapter introduced a transformational method for the development of multimodal user interfaces based on an introduced design space composed of design options. This transformational method will be applied in Chapter 5 on two case studies.

CHAPTER 5 CASE STUDIES

5.1 Introduction

After presenting a transformational method for the development of multimodal user interfaces based on design options introduced in Chapter 5, the current chapter aims at assessing the feasibility of the approach on two case studies. Section 5.2 illustrates a case study which concerns the development of an on-line polling system, while Section 5.3 presents the development of a car rental system. Each case study details the design and development of different versions of a user interface: monomodal or multimodal (graphical and vocal). Researches issued from user test [Meri06] conducted to assess the implementation of a system in monomodal and multimodal versions showed that multimodal user interfaces are preferred by the users as they ease the interaction with the system and make the applications more convivial.

5.2 Case study 1: Virtual Polling System

This case study describes a transformational approach for developing a UI concerning an opinion polling system aiming at collecting opinions of users regarding a certain subject. The scenario of this case study (Figure 5-1) is the following: from the Task and Domain Models, an AUI is produced, from which four CUIs are derived (graphical UI, vocal UI and two multimodal UIs with graphical and vocal predominance respectively). In the last step four FUIs are derived corresponding to each CUI obtained in the previous step.

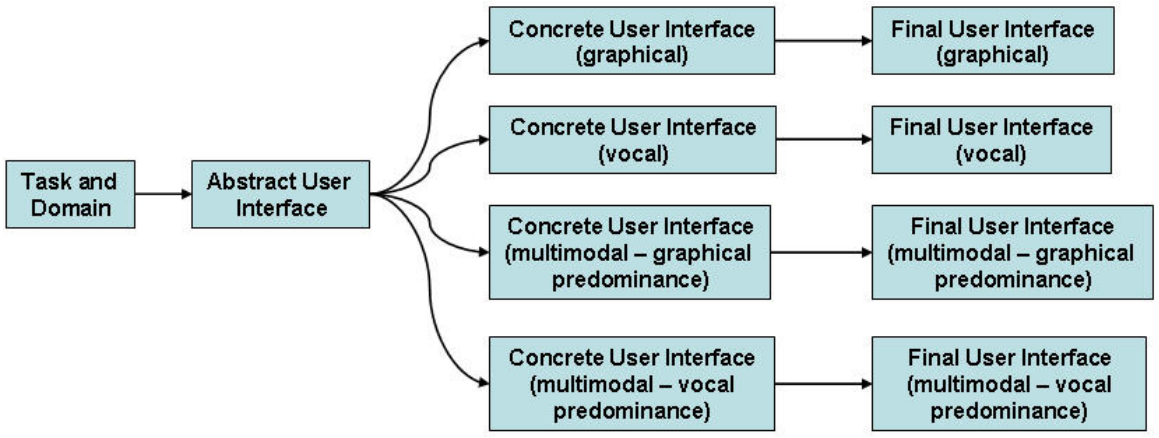

Task and Domain
Abstract User Interface
Concrete User Interface (graphical)
Concrete User Interface (vocal)
Concrete User Interface (multimodal – graphical predominance)
Concrete User Interface (multimodal – vocal predominance)
Final User Interface (graphical)
Final User Interface (vocal)
Final User Interface (multimodal – graphical predominance)
Final User Interface (multimodal – vocal predominance)

Figure 5-1. Development scenario for virtual polling system case study

5.2.1 Step 1: The Task and Domain Models

The task model, the domain model and the mappings between are graphically described using IdealXML, an Interface Development Environment for AppLications specified in UsiXML.

The upper part of Figure 5-2 depicts a CTT representation of the task model envisioned for the future system. The root task consists of participating to an opinion poll. In order to do this, the user has to provide the system with personal data like name, zip code, gender, age category. After that, the user iteratively answers some questions. Answering a question is composed of a system task showing the title of the question and of an interactive task consisting in selecting one answer among several proposed ones. Once the questions are answered, the questionnaire is sent back to its initiator. The bottom part of Figure 5-2 illustrates the domain model of our UI as produced by a software engineer. The domain model has the appearance of a class diagram and can be described as follow: a participant participates to a questionnaire, a questionnaire is made of several questions and a question is attached to a series of answers.

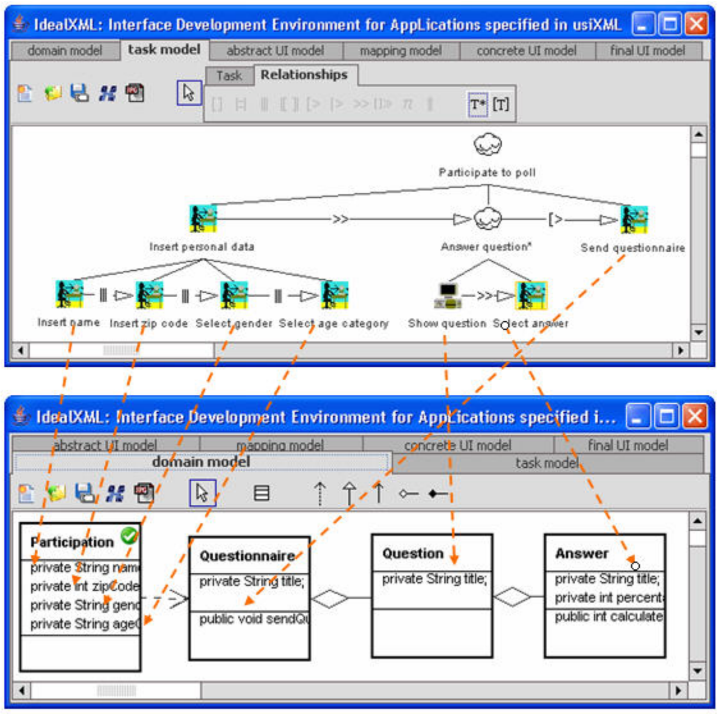
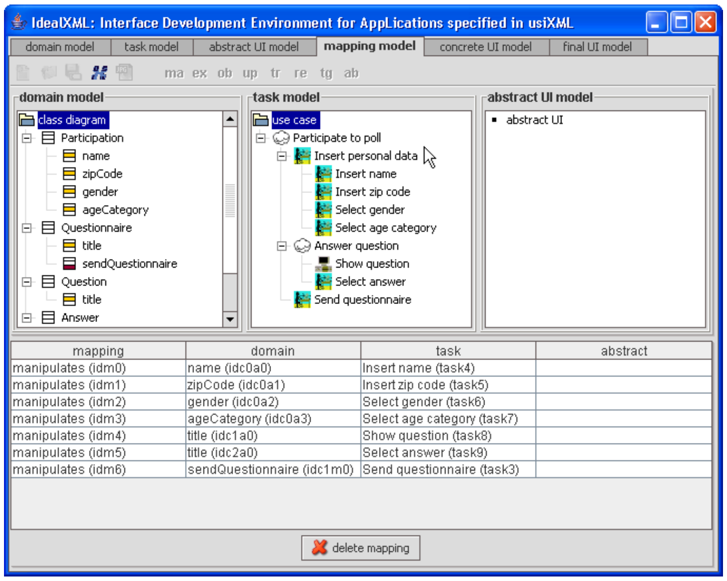


Figure 5-2. Mappings between the *Task Model* and the *Domain Model*

The dashed arrows between the two models in Fig. 5-2 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 (**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 5-3 is a screen shot of the IdealXML tool showing the graphical editor of the *Mapping Model*. Each leaf tasks is mapped on the corresponding attribute or method of the classes contained in the *Domain Model*.



mapping	domain	task	abstract
manipulates (idm0)	name (idc0a0)	Insert name (task4)	
manipulates (idm1)	zipCode (idc0a1)	Insert zip code (task5)	
manipulates (idm2)	gender (idc0a2)	Select gender (task6)	
manipulates (idm3)	ageCategory (idc0a3)	Select age category (task7)	
manipulates (idm4)	title (idc1a0)	Show question (task8)	
manipulates (idm5)	title (idc2a0)	Select answer (task9)	
manipulates (idm6)	sendQuestionnaire (idc1m0)	Send questionnaire (task3)	

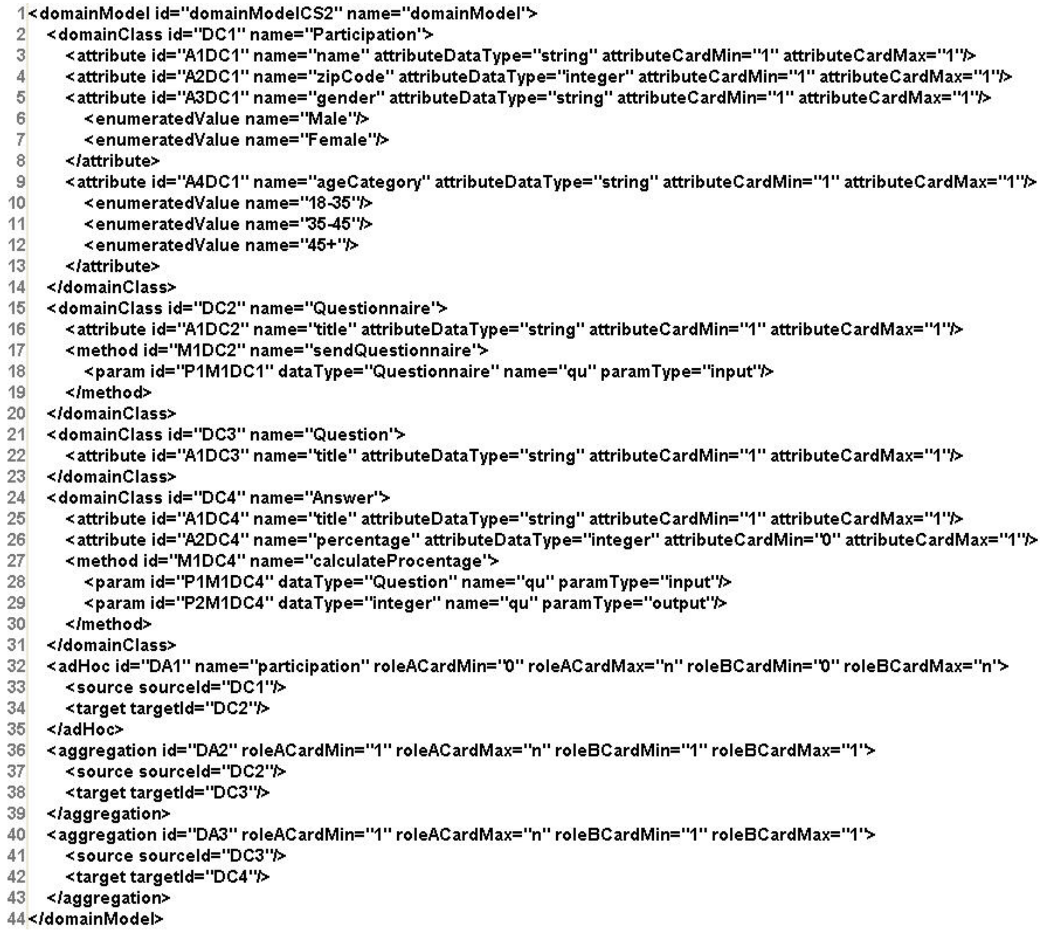
Figure 5-3. Mapping model for the virtual polling system

IdealXML generates automatically the UsiXML specifications for the *Task*, *Domain* and *Mapping Models*. Figure 5-4 describes the UsiXML specification corresponding to the *Task Model*. The first 14 lines describe the hierarchical decomposition of the *Task Model*, while lines 15 to 43 describe the relationships between the tasks.

```
 1<taskModel id="TaskModelCS1" name="TaskModel">
 2  <task id="Root" name="Participate to poll" importance="5" type="abstract">
 3    <task id="T1" name="Insert personal data" importance="3" type="interactive">
 4      <task id="T11" name="Insert name" userAction="create" taskItem="element" importance="5" type="interactive"/>
 5      <task id="T12" name="Insert zip code" userAction="create" taskItem="element" importance="5" type="interactive"/>
 6      <task id="T13" name="Select gender" userAction="select" taskItem="element" importance="5" type="interactive"/>
 7      <task id="T14" name="Select age category" userAction="select" taskItem="element" importance="5" type="interactive"/>
 8    </task>
 9    <task id="T2" name="Answer question" importance="3" type="abstract">
10      <task id="T21" name="Show question" userAction="create" taskItem="collection" importance="5" type="system"/>
11      <task id="T22" name="Select answer" userAction="select" taskItem="element" importance="5" type="interactive"/>
12    </task>
13    <task id="T3" name="Send questionnaire" userAction="start" taskItem="operation" importance="3" type="interactive"/>
14  </task>
15  <enabling id="e1">
16    <source sourceId="T1"/>
17    <target targetId="T2"/>
18  </enabling>
19  <disabling id="e2">
20    <source sourceId="T2"/>
21    <target targetId="T3"/>
22  </disabling>
23  <independentConcurrency id="e11">
24    <source sourceId="T11"/>
25    <target targetId="T12"/>
26  </independentConcurrency>
27  <independentConcurrency id="e12">
28    <source sourceId="T12"/>
29    <target targetId="T13"/>
30  </independentConcurrency>
31  <independentConcurrency id="e13">
32    <source sourceId="T13"/>
33    <target targetId="T14"/>
34  </independentConcurrency>
35  <iteration id="e3">
36    <source sourceId="T2"/>
37    <target targetId="T2"/>
38  </iteration>
39  <enabling id="e21">
40    <source sourceId="T21"/>
41    <target targetId="T22"/>
42  </enabling>
43</taskModel>
```

Figure 5-4. Task Model expressed in UsiXML

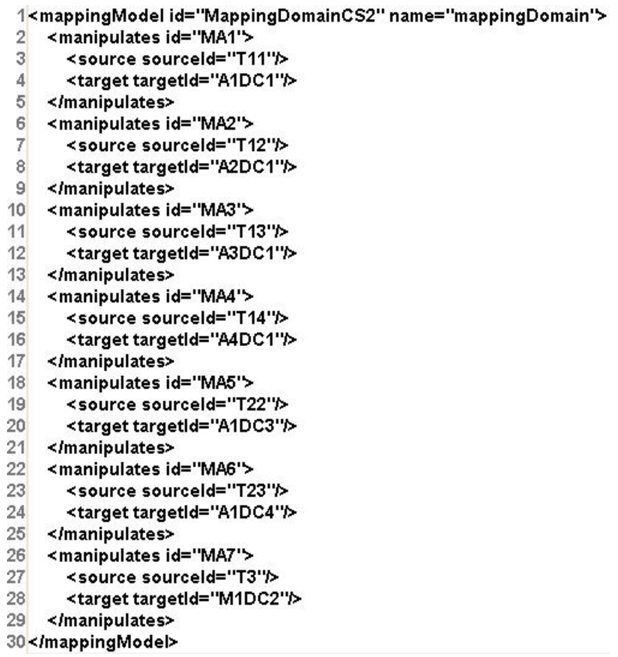
Figure 5-5 illustrates the *Domain Model* expressed in UsiXML. Lines 1 to 31 define the classes that are involved into the class diagram. Lines 9 to 12 describe the attribute "ageCategory" that can have different values expressed under the form of an enumerated domain (the possible values are "18-35", "35-45", "more then 45"). Lines 27 to 30 define a method with its two parameters, an input parameter and an output parameter. Lines 32 to 44 describe the relationships between the above described classes.



```
 1<domainModel id="domainModelCS2" name="domainModel">
 2  <domainClass id="DC1" name="Participation">
 3    <attribute id="A1DC1" name="name" attributeDataType="string" attributeCardMin="1" attributeCardMax="1"/>
 4    <attribute id="A2DC1" name="zipCode" attributeDataType="integer" attributeCardMin="1" attributeCardMax="1"/>
 5    <attribute id="A3DC1" name="gender" attributeDataType="string" attributeCardMin="1" attributeCardMax="1"/>
 6      <enumeratedValue name="Male"/>
 7      <enumeratedValue name="Female"/>
 8    </attribute>
 9    <attribute id="A4DC1" name="ageCategory" attributeDataType="string" attributeCardMin="1" attributeCardMax="1"/>
10      <enumeratedValue name="18-35"/>
11      <enumeratedValue name="35-45"/>
12      <enumeratedValue name="45+"/>
13    </attribute>
14  </domainClass>
15  <domainClass id="DC2" name="Questionnaire">
16    <attribute id="A1DC2" name="title" attributeDataType="string" attributeCardMin="1" attributeCardMax="1"/>
17    <method id="M1DC2" name="sendQuestionnaire">
18      <param id="P1M1DC1" dataType="Questionnaire" name="qu" paramType="input"/>
19    </method>
20  </domainClass>
21  <domainClass id="DC3" name="Question">
22    <attribute id="A1DC3" name="title" attributeDataType="string" attributeCardMin="1" attributeCardMax="1"/>
23  </domainClass>
24  <domainClass id="DC4" name="Answer">
25    <attribute id="A1DC4" name="title" attributeDataType="string" attributeCardMin="1" attributeCardMax="1"/>
26    <attribute id="A2DC4" name="percentage" attributeDataType="integer" attributeCardMin="0" attributeCardMax="1"/>
27    <method id="M1DC4" name="calculateProcentage">
28      <param id="P1M1DC4" dataType="Question" name="qu" paramType="input"/>
29      <param id="P2M1DC4" dataType="integer" name="qu" paramType="output"/>
30    </method>
31  </domainClass>
32  <adHoc id="DA1" name="participation" roleACardMin="0" roleACardMax="n" roleBCardMin="0" roleBCardMax="n">
33    <source sourceId="DC1"/>
34    <target targetId="DC2"/>
35  </adHoc>
36  <aggregation id="DA2" roleACardMin="1" roleACardMax="n" roleBCardMin="1" roleBCardMax="1">
37    <source sourceId="DC2"/>
38    <target targetId="DC3"/>
39  </aggregation>
40  <aggregation id="DA3" roleACardMin="1" roleACardMax="n" roleBCardMin="1" roleBCardMax="1">
41    <source sourceId="DC3"/>
42    <target targetId="DC4"/>
43  </aggregation>
44</domainModel>
```

Figure 5-5. Domain Model expressed in UsiXML

Figure 5-6 illustrates the mappings established between the *Task Model* and the *Domain Model*. These mappings are specified in UsiXML with the use of two tags (i.e., <source> and <target>) that identify which task will manipulate which attribute/method from the domain model.



```
<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 5-6. Mapping Model expressed in UsiXML

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

The second transformation step involves a transformation system that contains rules applied in order to realize the transition from the task and domain model to the abstract model. The rules contained in this transformation system have been designed independently of the types of concrete UI (i.e., graphical, vocal or multimodal) that will be obtained in the next step. This step is subdivided into five sub-steps according to [Limb04b]. We are improving this work by offering the complete set of rules and by adapting them to the needs of a multimodal UI.

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

The AUI structure is obtained by applying the rules described in Figures 5-7, 5-8, 5-9, 5-10 and 5-11. The result of the application of these rules over the task model structure consists in a hierarchical decomposition of the AUI into abstract containers and abstract individual components.

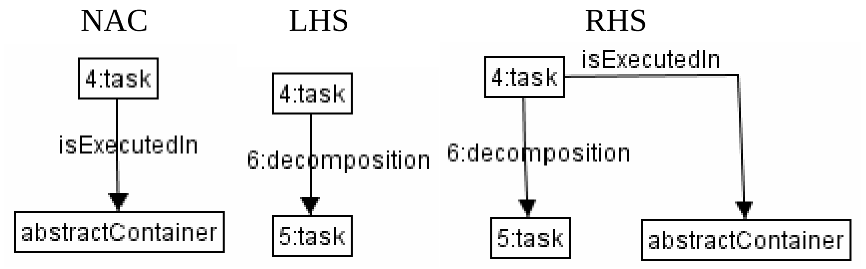


Figure 5-7. Create an AC for task that has task children

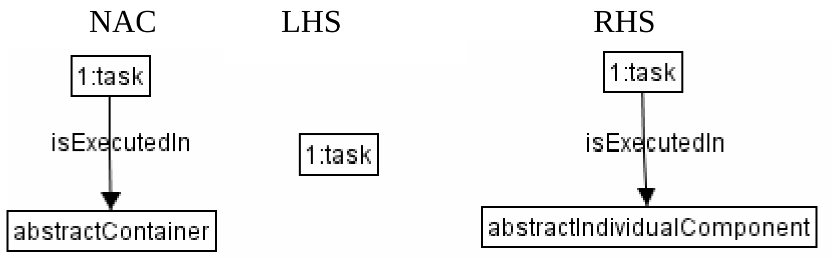


Figure 5-8. Create an AIC for leaf tasks

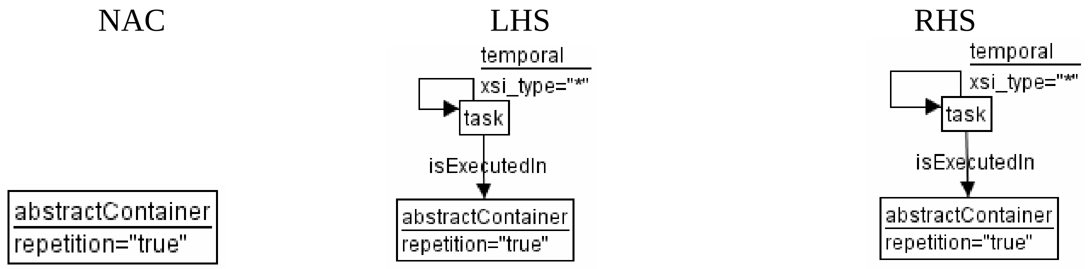


Figure 5-9. Iterative tasks are mapped onto repetitive AC

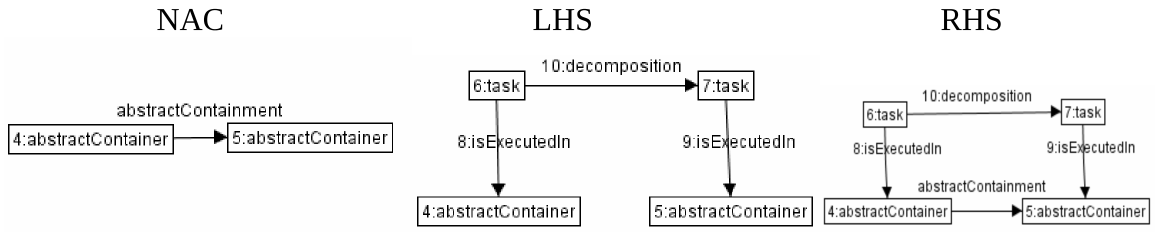


Figure 5-10. Reconstruct containment relationship between ACs

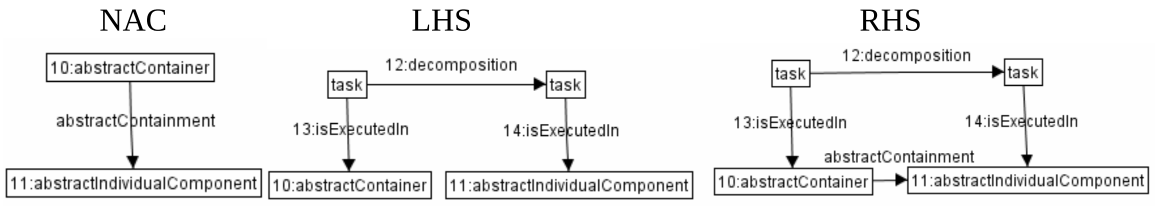


Figure 5-11. Reconstruct containment relationship between ACs and AICs

Sub-step 2.2: Rules for the selection of AICs

Depending on the mappings between the *Task Model* and the *Domain Model*, different facets will be created for the AICs that support the execution of the task. We have redefined the possible values for the attributes of the facet component as described in Chapter 4. Accordingly, the following facets will be created:

- Input facet of type create element for the AIC that are assigned to the execution of the following tasks: **create name** and **create zipCode** (Figure 5-12)
- Input facet of type select element for the AIC that are assigned to the execution of the following tasks: **select gender, select ageCategory** and **select Answer** (Figure 5-13); 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 (Figure 5-14)

- Output facet of type convey element for the AIC assigned to the task **Show Question Title** (Figure 5-15)
- Control facet of type start operation for the AIC dedicated to the task **Send Questionnaire** (Figure 5-16)

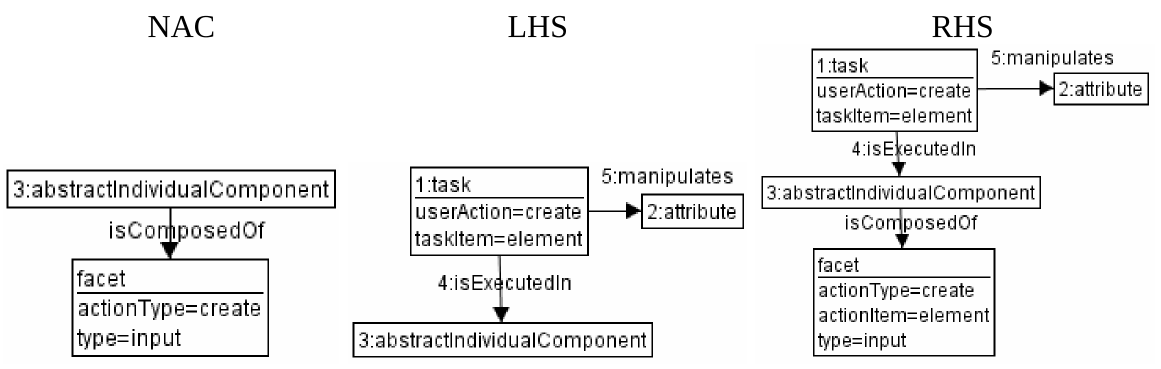


Figure 5-12. Create an input facet for AIC executed in tasks of type create

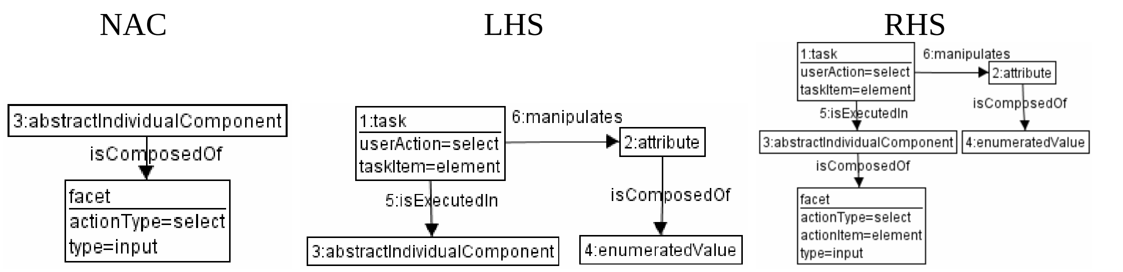


Figure 5-13. Create an input facet of type select element when an enumerated value attribute is encountered

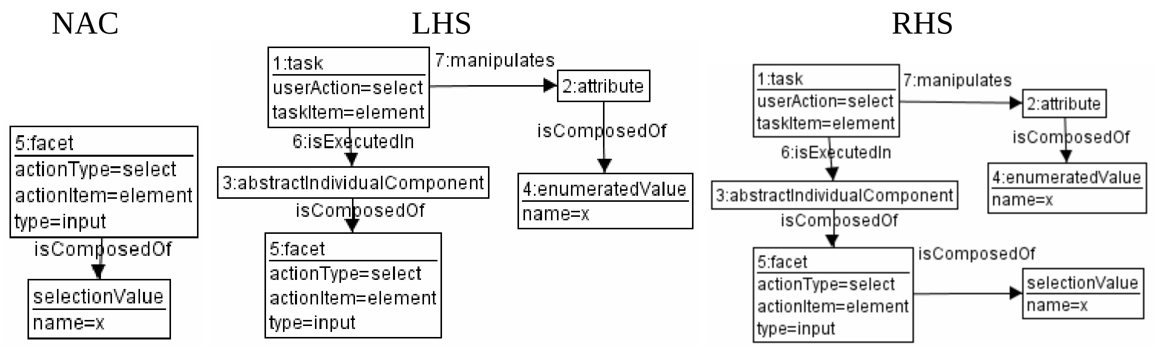


Figure 5-14. Create selection values for facets of type select for each enumerated value of an attribute

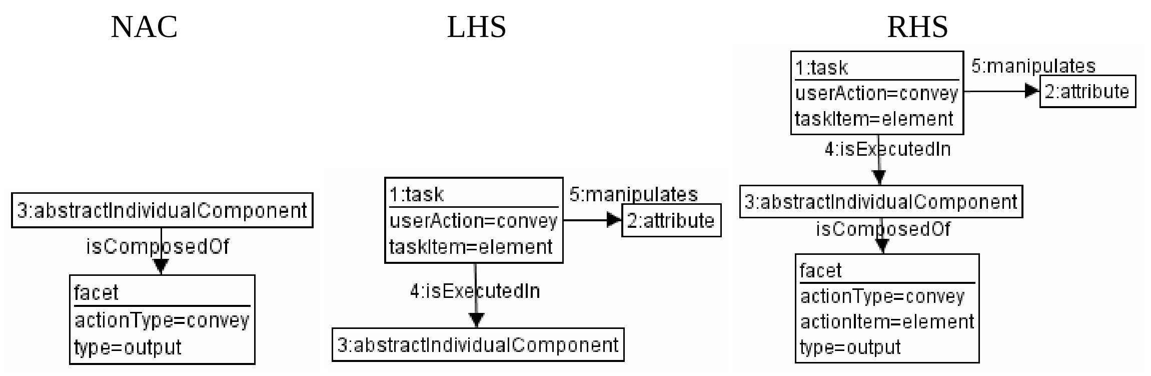


Figure 5-15. Create an output facet that conveys an element

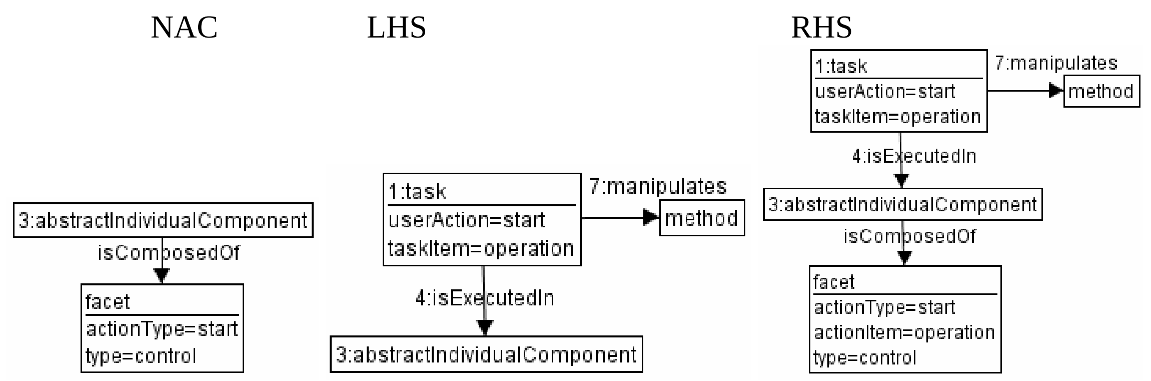
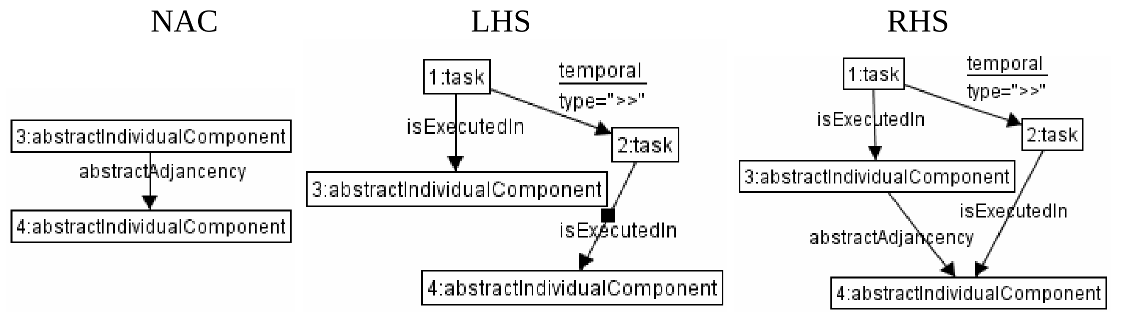


Figure 5-16. Create a control facet of type start operation when a method is manipulated by a task

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

For each couple of sister tasks executed into AIOs, we define an abstractAdjacency relationship between these AIOs. As AIOs can be of two types (i.e., ACs or AICs), there are four possible combination to take into account. For each combination a specific rule is considered (Figure 5-17 and Figures B-1, B-2 and B-3 in Annex B).


NAC
LHS
RHS
3:abstractIndividualComponent
abstractAdjacency
4:abstractIndividualComponent
1:task
temporal
type=">>"
2:task
isExecutedIn
3:abstractIndividualComponent
isExecutedIn
4:abstractIndividualComponent
1:task
temporal
type=">>"
2:task
isExecutedIn
3:abstractIndividualComponent
isExecutedIn
abstractAdjacency
4:abstractIndividualComponent
Figure 5-17. Creating abstract adjacency for <AIC, AIC> couple

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

In a similar way as the previous sub-step, for each couple of sister tasks executed into AIOs, we define an abstractDialogControl relationship between these AIOs 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 combination to take into account. For each combination a specific rule is considered (Figure 5-18 and Figures B-4, B-5 and B-6 in Annex B).

NAC LHS RHS

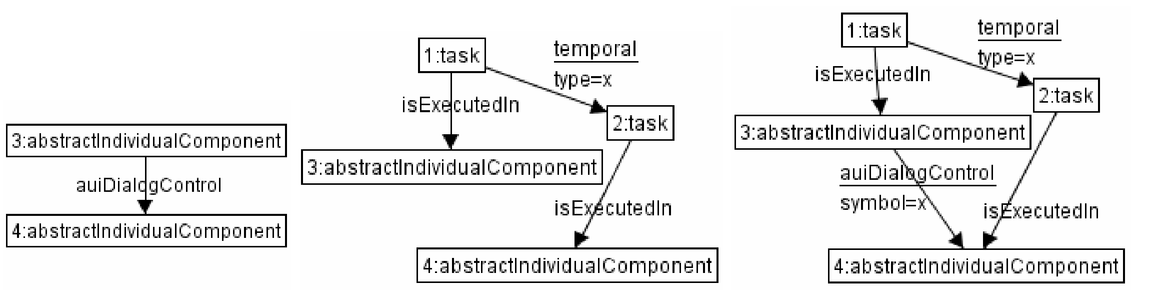

3:abstractIndividualComponent
auiDialogControl
4:abstractIndividualComponent
1:task
temporal
type=x
isExecutedIn
2:task
3:abstractIndividualComponent
isExecutedIn
4:abstractIndividualComponent
1:task
temporal
type=x
isExecutedIn
2:task
3:abstractIndividualComponent
auiDialogControl
symbol=x
isExecutedIn
4:abstractIndividualComponent

Figure 5-18. Deriving Abstract Dialog Control for <AIC, AIC> couple

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

Figure 5-19 illustrates the rule used to synchronize the AICs with the attribute of an object from the Domain Model. This synchronization is done through the **updates** mapping relationship.

NAC LHS RHS

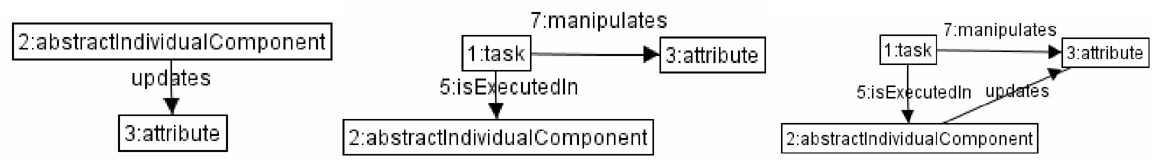


Figure 5-19. Deriving updates relationships for an AIC

In order to allow the triggering of a method by an AIC, a mapping relationship of type **triggers** is generated by the rule described in Figure 5-20.

NAC LHS RHS

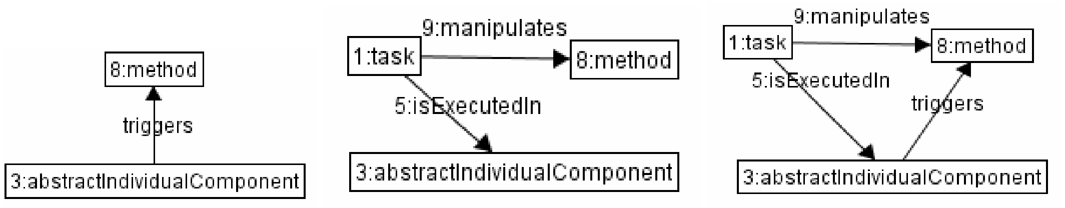


Figure 5-20. Deriving trigger relationships for AICs

The UsiXML specification corresponding to the AUI Model can be obtained by following two different directions:

- Executing the transformation rules according to the above described sub-steps, which is a machine processable approach

- Designing graphically the AUI Model within the TransformiXML tool and generating the corresponding specification , which makes it a human readable approach.

As the main objective of UsiXML is to provide first a machine processable language and then a human readable specification, the AUI Model of the virtual polling system is obtained by following the first direction. However, we illustrate in Figure 5-21 the representation of the AUI Model within IdealXML tool.

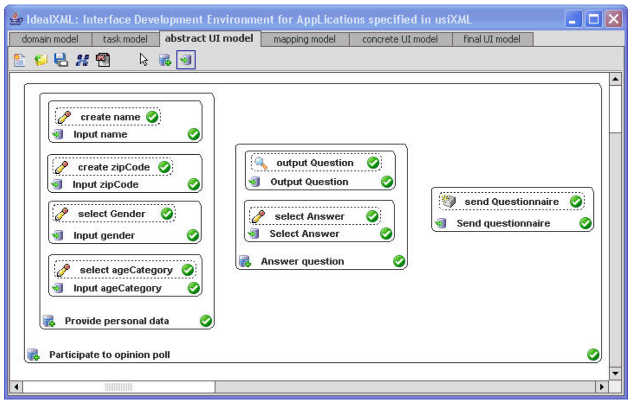
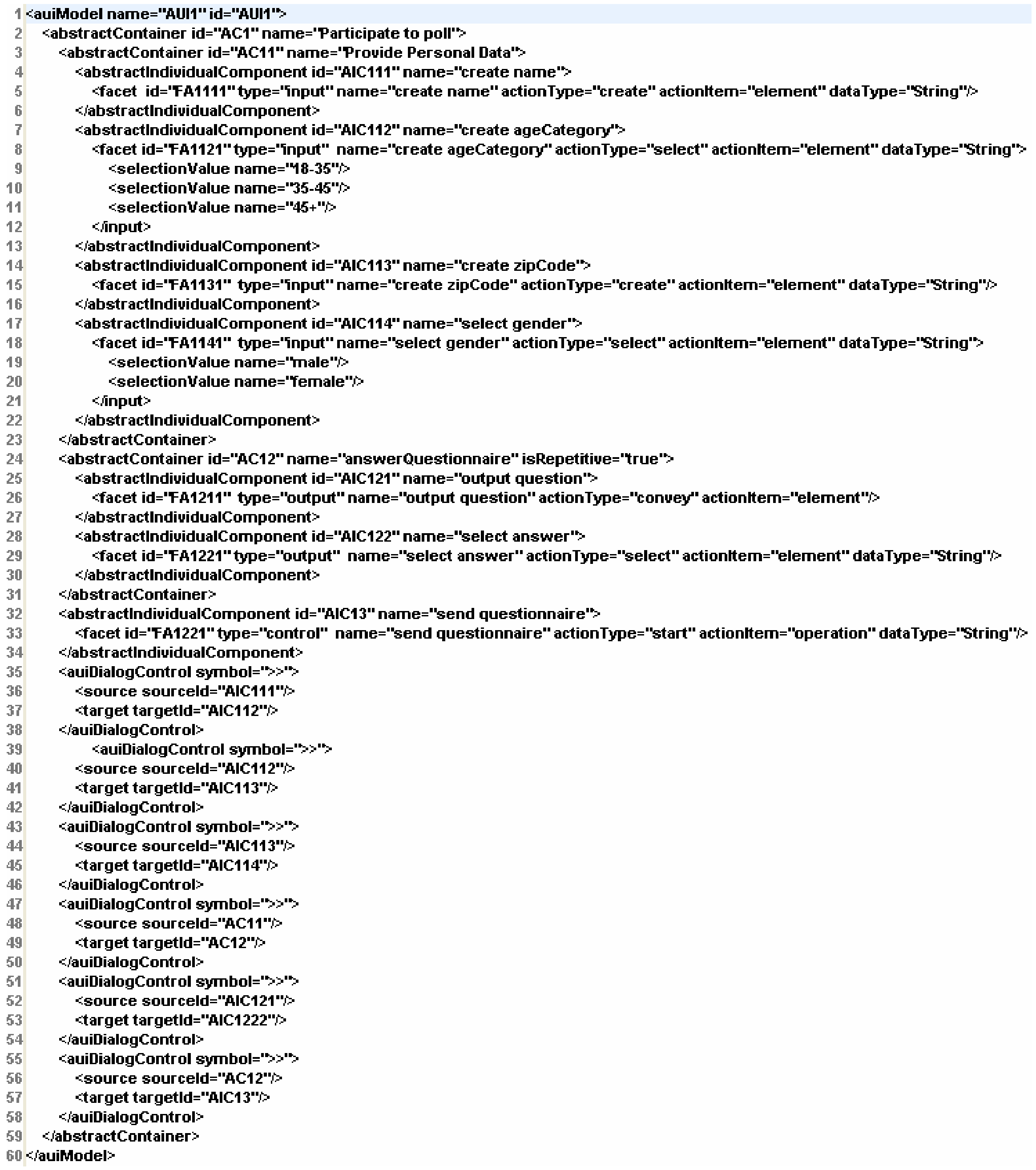


Figure 5-21 AUI Model of virtual polling system designed in TransformiXML

The resultant UsiXML specification is shown in Figure 5-22. Lines 1 to 34 define the AIOs of the virtual polling system, while lines 35 to 60 specify the dialog control relationships between them.



```
<auiModel 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+"/>
        </input>
      </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"/>
        </input>
      </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>
    <auiDialogControl symbol=">>">
      <source sourceId="AIC111"/>
      <target targetId="AIC112"/>
    </auiDialogControl>
        <auiDialogControl symbol=">>">
      <source sourceId="AIC112"/>
      <target targetId="AIC113"/>
    </auiDialogControl>
    <auiDialogControl symbol=">>">
      <source sourceId="AIC113"/>
      <target targetId="AIC114"/>
    </auiDialogControl>
    <auiDialogControl symbol=">>">
      <source sourceId="AC11"/>
      <target targetId="AC12"/>
    </auiDialogControl>
    <auiDialogControl symbol=">>">
      <source sourceId="AIC121"/>
      <target targetId="AIC1222"/>
    </auiDialogControl>
    <auiDialogControl symbol=">>">
      <source sourceId="AC12"/>
      <target targetId="AIC13"/>
    </auiDialogControl>
  </abstractContainer>
</auiModel>
```

Figure 5-22. AUI Model expressed in UsiXML

5.2.3 Step 3: From AUI Model to CUI Model

The third step implies a transformational system that is composed of necessary rules for realizing the transition from AUI to CUIs. Four CUI are taken into account:

1. **Case 1 - graphical UI:** the modality used to interact with the system is entirely graphical (monomodal UI)
2. **Case 2- vocal UI:** the modality used to interact with the system is entirely vocal (monomodal UI)

3. **Case 3 - multimodal UI with graphical predominance:** in order to fulfill his task the user employs the graphical interaction in a higher proportion than the vocal interaction
4. **Case 4 - multimodal UI with vocal predominance:** in order to fulfill his task the user employs the vocal interaction in a higher proportion than the graphical interaction

Design decisions concerning the type of interaction that is the most suitable for different tasks in a user interface where assessed by different research studies between which [Meri06]. These studies show that the graphical interaction is usually preferred by end-users while visualizing the output data (they have the possibility of rereading them) and submitting input data. The vocal interaction is mostly used in the case of a precise and short input data, such as for answering the questionnaire.

For each type of CUI defined above, a transformation system containing specific rules is designed. In the following it will be emphasized the modularity and the extensibility of the transformation rules applied in order to obtain the desired CUIs.

Case 1: generation of graphical UI

Sub-step 3.1: Reification of AC into CC

The rule described in Figure 5-23 creates a GC which will be the **main box** of the UI associated to the AC found one level under the root AC in the abstract hierarchy. This **main box** is embedded into the main window of the UI.

The rule described in Figure 5-24 creates a GC of type **box** for each AC contained into an AC that was reified into a **main box**.

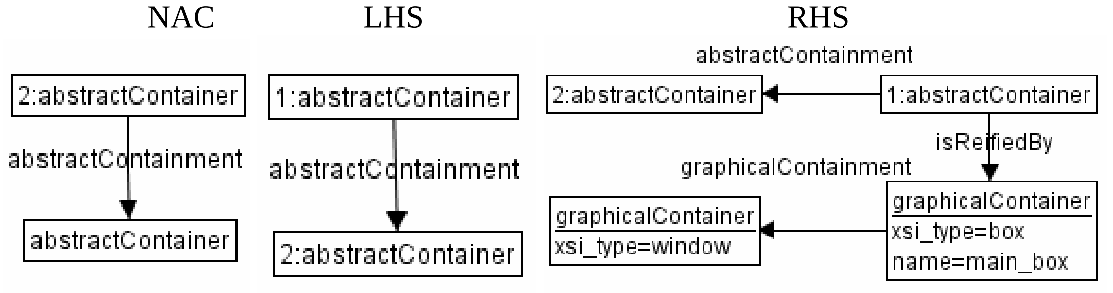


Figure 5-23. Creation of windows derived from abstract containment relationship

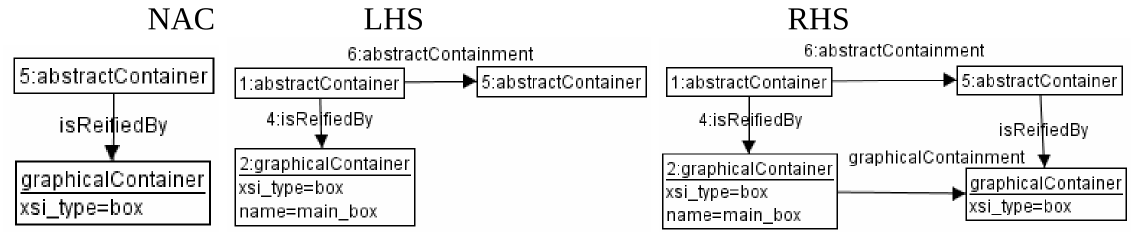


Figure 5-24. Generation of graphical containers of type box

Sub-step 3.2: Selection of CICs

The rule illustrated in Figure 5-25 generates a GC of type **box** that will embed two GICs: an **outputText** and an **inputText** representing respectively the label and the associated text field. These GICs are used to insert the **name** and the **zipCode**. The rule is applied each time when an AIC with an input facet of type create element is encountered.

Figure 5-26 describes the rule applied in order to create a GC of type **box** that will embed a **group of radio buttons** when an input facet of type select element is encountered. A GIC of type **outputText** representing the label associated to this group is also created. The **defaultContent** of the GIC is the same as the name of the AIC. The radio buttons associated to this group are created by executing the rule described in Figure 5-27. For each selection value of a facet of type select, a radio button, that has the same content as the name of the selection value, is created. These two last rules are used in order to select the **gender** of the user, the **ageCategory** and also his **answers** to the questions.

NAC LHS RHS

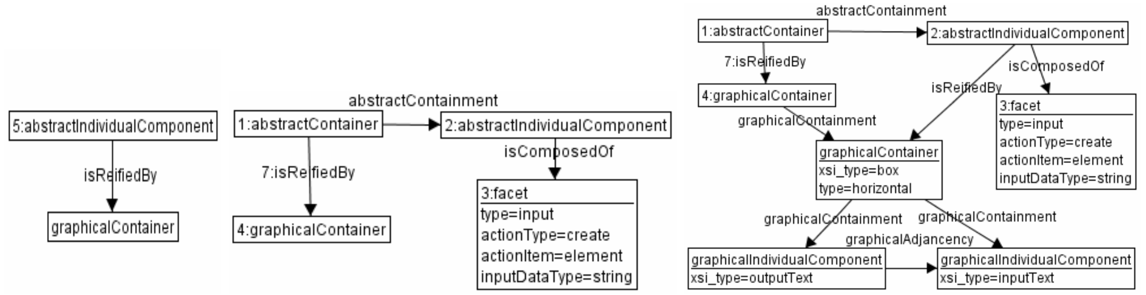

5:abstractIndividualComponent
isReifiedBy
graphicalContainer
1:abstractContainer
abstractContainment
2:abstractIndividualComponent
7:isReifiedBy
isComposedOf
4:graphicalContainer
3:facet type=input actionType=create actionItem=element inputDataType=string
graphicalContainment
graphicalContainer xsi_type=box type=horizontal
graphicalAdjancency
graphicalIndividualComponent xsi_type=outputText
graphicalIndividualComponent xsi_type=inputText

Figure 5-25. Generation of an outputText and an inputText for AIC with create input facet

NAC LHS RHS

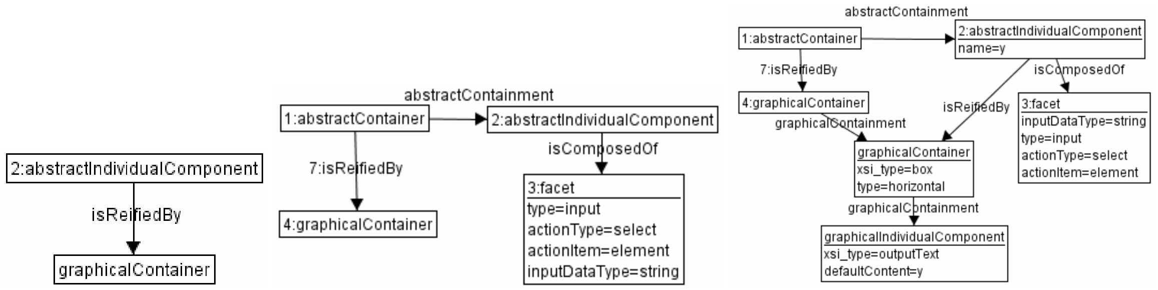

2:abstractIndividualComponent
isReifiedBy
graphicalContainer
1:abstractContainer
abstractContainment
2:abstractIndividualComponent name=y
7:isReifiedBy
isComposedOf
4:graphicalContainer
3:facet type=input actionType=select actionItem=element inputDataType=string
graphicalContainment
graphicalContainer xsi_type=box type=horizontal
graphicalIndividualComponent xsi_type=outputText defaultContent=y

Figure 5-26. Generation of a graphical container of type box that will contain a group of radio buttons

NAC LHS RHS

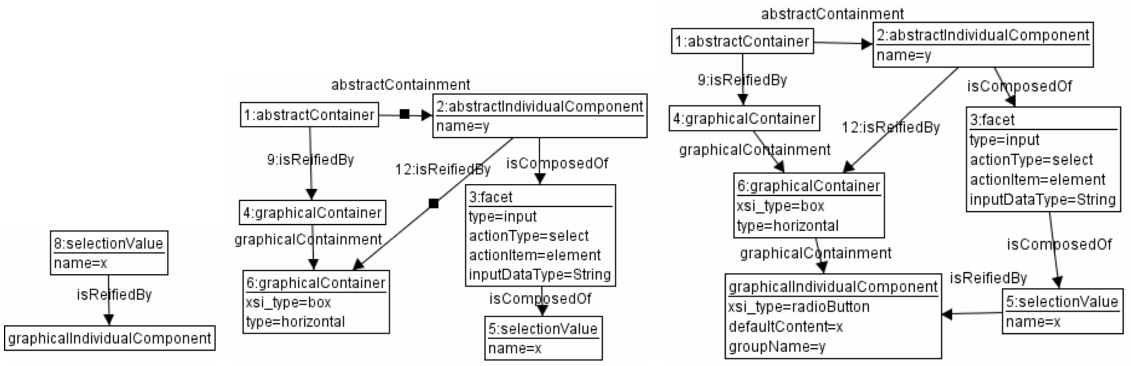

8:selectionValue name=x
isReifiedBy
graphicalIndividualComponent
1:abstractContainer
abstractContainment
2:abstractIndividualComponent name=y
9:isReifiedBy
12:isReifiedBy
isComposedOf
4:graphicalContainer
graphicalContainment
6:graphicalContainer xsi_type=box type=horizontal
3:facet type=input actionType=select actionItem=element inputDataType=String
5:selectionValue name=x
isReifiedBy
graphicalIndividualComponent xsi_type=radioButton defaultContent=x groupName=y

Figure 5-27. Generation of radioButtons for each selection value of a facet of type select

Figure 5-28 presents the rule applied to generate a GC of type **outputText**, each time when an output facet of type create is encountered. Thus, the **titles of the questions** are created.

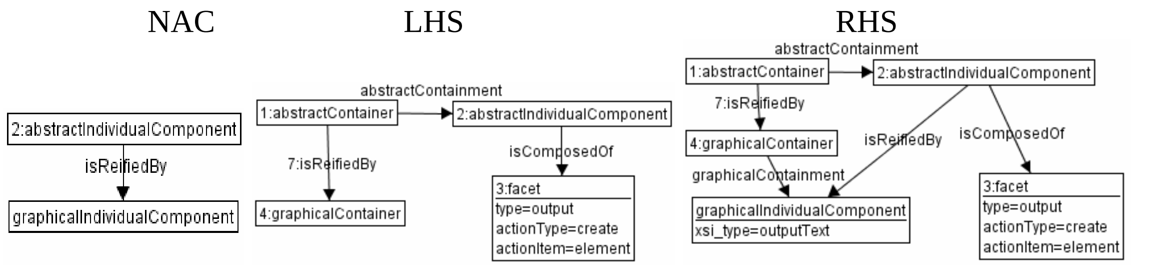


Figure 5-28. Generation of an outputText for an output facet with create action type

In order to generate the button that will allow users to **send the questionnaire,** rule presented in Figure 5-29 was designed. The GIC of type **button** is created when a control facet of type start operation is encountered.

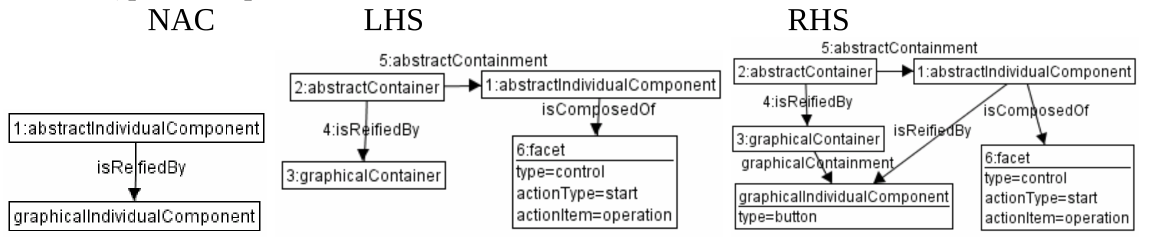


Figure 5-29. Generation of a control button

Sub-step 3.3: Arrangement of CICs

For each couple of adjacent AIOs that are reified into graphicalCIOs, we define a graphicalAdjencency relationship between these graphicalCIOs. As AIOs can be of two types (i.e., ACs or AICs), there are four possible combination to take into account. For each combination a specific rule is considered (Figure 5-30 and Figures B-7, B-8 and B-9 in Annex B).

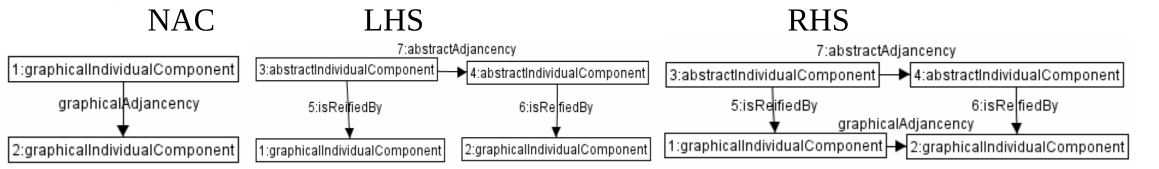

NAC
LHS
RHS
1:graphicalIndividualComponent
graphicalAdjancency
2:graphicalIndividualComponent
7:abstractAdjancency
3:abstractIndividualComponent
4:abstractIndividualComponent
5:isReifiedBy
6:isReifiedBy
1:graphicalIndividualComponent
2:graphicalIndividualComponent
7:abstractAdjancency
3:abstractIndividualComponent
4:abstractIndividualComponent
5:isReifiedBy
6:isReifiedBy
graphicalAdjancency
1:graphicalIndividualComponent
2:graphicalIndividualComponent

Figure 5-30. Generation of Graphical Adjacency relationships for <GIC, GIC> couples

Sub-step 3.4: Navigation definition

The rules that ensure the navigation definition are not exemplified for this case study as all the components of the virtual polling system are presented into the same window. For a complete set of rules defining the navigation definition based on design options, please refer to Section 5.3.

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 are of two types (i.e., ACs and AICs), four rules describing the four possible combinations are considered (Figure 5-31 and Figures B-10, B-11 and B-12 in Annex B).

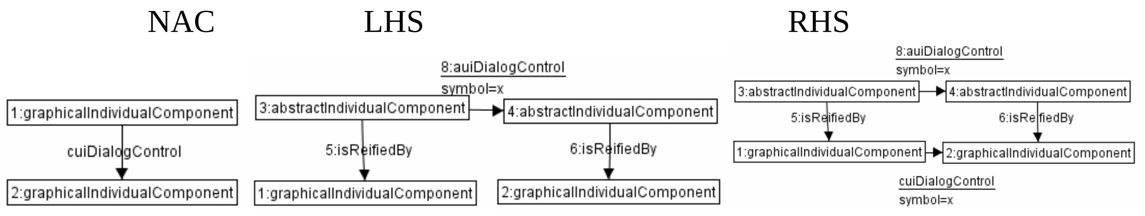

NAC
LHS
RHS
1:graphicalIndividualComponent
cuiDialogControl
2:graphicalIndividualComponent
8:auiDialogControl
symbol=x
3:abstractIndividualComponent
4:abstractIndividualComponent
5:isReifiedBy
6:isReifiedBy
1:graphicalIndividualComponent
2:graphicalIndividualComponent
8:auiDialogControl
symbol=x
3:abstractIndividualComponent
4:abstractIndividualComponent
5:isReifiedBy
6:isReifiedBy
1:graphicalIndividualComponent
2:graphicalIndividualComponent
cuiDialogControl
symbol=x

Figure 5-31. Generation of Concrete Dialog Control relationships for <GIC, GIC> couples

Sub-step 3.6: Derivation of CUI to Domain Relationship

Figure 5-32 illustrates the rule used to map GICs with the corresponding attribute of an object from the Domain Model. The **updates** relationship is transposed from the AIC that is reified by the GIC.

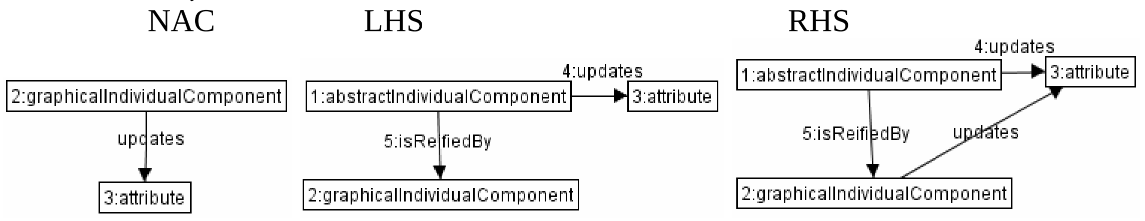

NAC
LHS
RHS
2:graphicalIndividualComponent
updates
3:attribute
1:abstractIndividualComponent
4:updates
3:attribute
5:isReifiedBy
2:graphicalIndividualComponent
4:updates
1:abstractIndividualComponent
3:attribute
5:isReifiedBy
updates
2:graphicalIndividualComponent

Figure 5-32. Transposition of update relationship

Figure 5-33 illustrates the rule used to map GICs with the corresponding method of an object from the Domain Model. The **triggers** relationship is transposed from the AIC that is reified by the GIC.

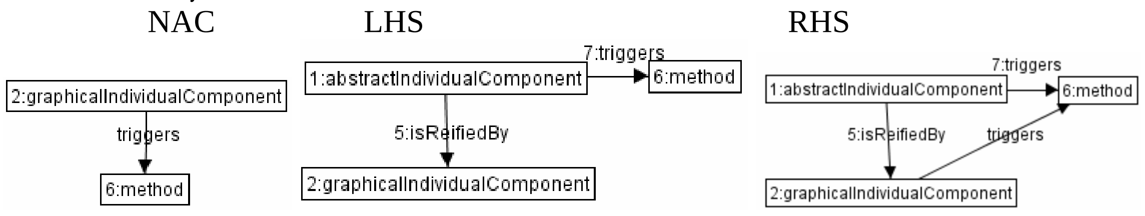

NAC
LHS
RHS
2:graphicalIndividualComponent
triggers
6:method
7:triggers
1:abstractIndividualComponent
6:method
5:isReifiedBy
2:graphicalIndividualComponent
7:triggers
1:abstractIndividualComponent
6:method
5:isReifiedBy
triggers
2:graphicalIndividualComponent

Figure 5-33. Transposition of triggers relationship

Case 2: generation of vocal UI

Sub-step 3.1: Reification of AC into CC

The rule described in Figure 5-34 creates a VC of type **vocalGroup** that will group all the vocal forms of the vocal application.

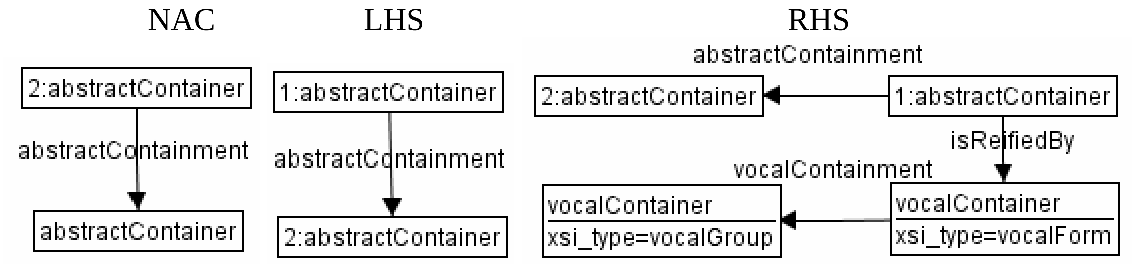


Figure 5-34. Generation of vocal containers

Figure 5-35 illustrates the rule applied in order to create a VC of type **vocalForm** for each AC contained into the AC that was reified in the rule above by the VC of type **vocalForm**.

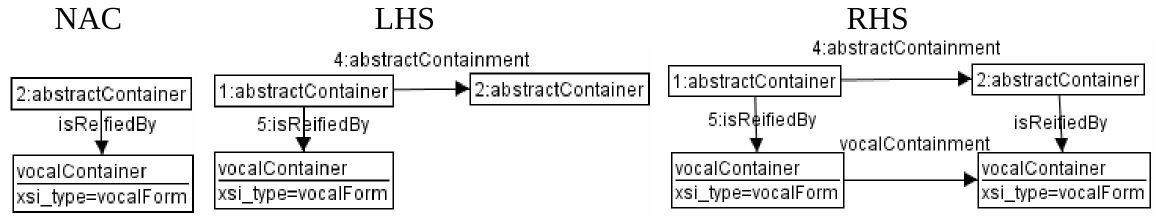


Figure 5-35. Generation of vocal containers of type vocalForm

Sub-step 3.2: Selection of CICs

The rule illustrated in Figure 5-36 is designed so as to allow users to introduce there **names**. The VC of type **vocalForm** that is generated contains the following VICs:

- **VocalPrompt**: the system is inviting the user to utter his name
- **VocalInput:** the user is uttering his name
- **VocalConfirmation** (with vocalPrompt and vocalInput)**:** the system is asking for the confirmation of the recognized input.

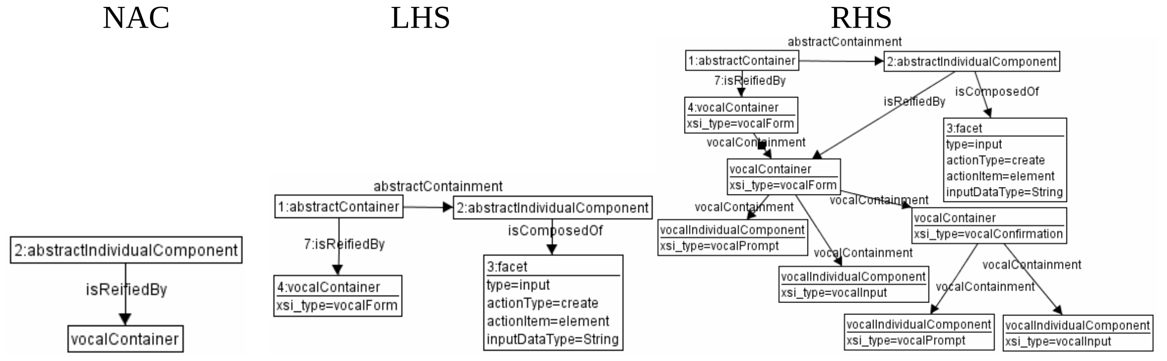
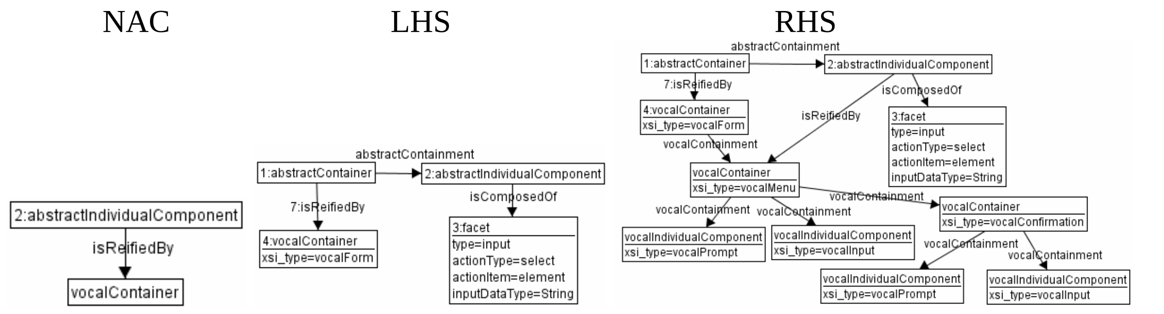
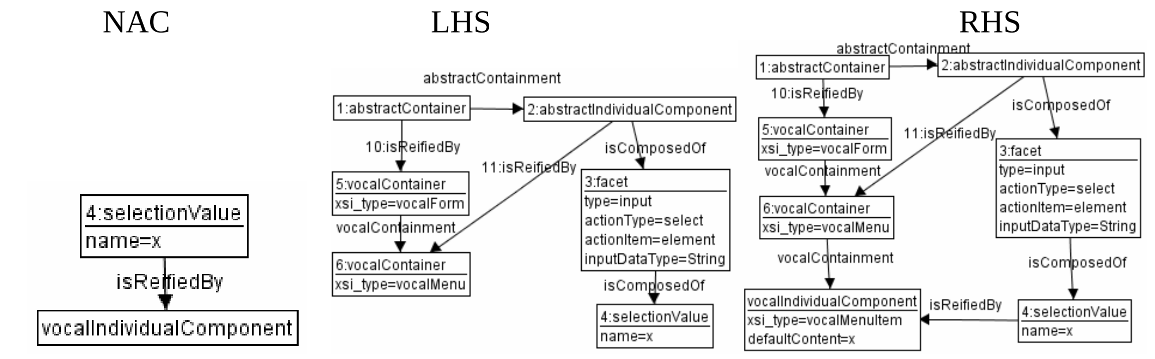


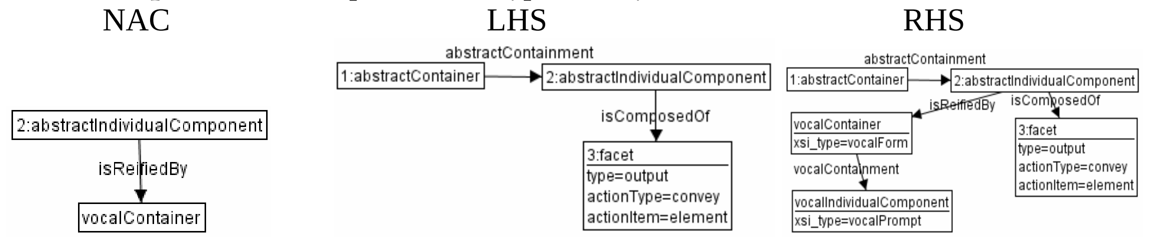
Figure 5-36. Generation of vocal containers of type vocal form that will contain a vocal prompt, a vocal input and a vocal container of type vocal confirmation

Figure 5-37 illustrates the rule used to create a VC of type vocalMenu containg a vocalPrompt, a vocalInput and a vocalConfirmation when an input facet of type select is encountered. The rule allows user to select his **gender**, his **age category** and the **answers to the questions.** In order to add the corresponding menu items for each selection value of the facet, the rule described in Figure 5-38 was designed. Each menu item will have as content the name of the selection value.


NAC
LHS
RHS
2:abstractIndividualComponent
isReifiedBy
vocalContainer
abstractContainment
1:abstractContainer
2:abstractIndividualComponent
7:isReifiedBy
isComposedOf
4:vocalContainer xsi_type=vocalForm
3:facet type=input actionType=select actionItem=element inputDataType=String
vocalContainer xsi_type=vocalMenu
vocalContainer xsi_type=vocalConfirmation
vocalIndividualComponent xsi_type=vocalPrompt
vocalIndividualComponent xsi_type=vocalInput
vocalContainment
Figure 5-37. Generation of vocal containers of type vocal menu when a select facet of an AIC is found


NAC
LHS
RHS
4:selectionValue name=x
isReifiedBy
vocalIndividualComponent
abstractContainment
1:abstractContainer
2:abstractIndividualComponent
10:isReifiedBy
11:isReifiedBy
isComposedOf
5:vocalContainer xsi_type=vocalForm
6:vocalContainer xsi_type=vocalMenu
vocalContainment
3:facet type=input actionType=select actionItem=element inputDataType=String
4:selectionValue name=x
vocalIndividualComponent xsi_type=vocalMenuItem defaultContent=x
Figure 5-38. Generation of vocal menu items for each selection value of AUI with a facet of type select

In order to announce the beginning of the **questionnaire section** the rule described in Figure 5-39 creates a VC of type **vocalForm** that will contain a VIC of type **vocalPrompt,** when an output facet of type convey element is identified.


NAC
LHS
RHS
2:abstractIndividualComponent
isReifiedBy
vocalContainer
abstractContainment
1:abstractContainer
2:abstractIndividualComponent
isComposedOf
3:facet type=output actionType=convey actionItem=element
vocalContainer xsi_type=vocalForm
vocalContainment
vocalIndividualComponent xsi_type=vocalPrompt
Figure 5-39. Generation of a vocalForm that contains a VIC of type vocalPrompt when an output facet of type convey element is encountered

The rule described in Figure 5-40 is designed in order to allow users to **send the questionnaire** and consists of a VC of type vocalForm that contains:

- **vocalPrompt:** the system asks the user whether he wants to send the questionnaire or not
- **vocalInput:** the user is uttering his answer
- **vocalConfirmation (with vocalPrompt and vocalInput):** the system solicits a confirmation of the recognized input

The vocal components are created when an output facet of type convey element is identified.

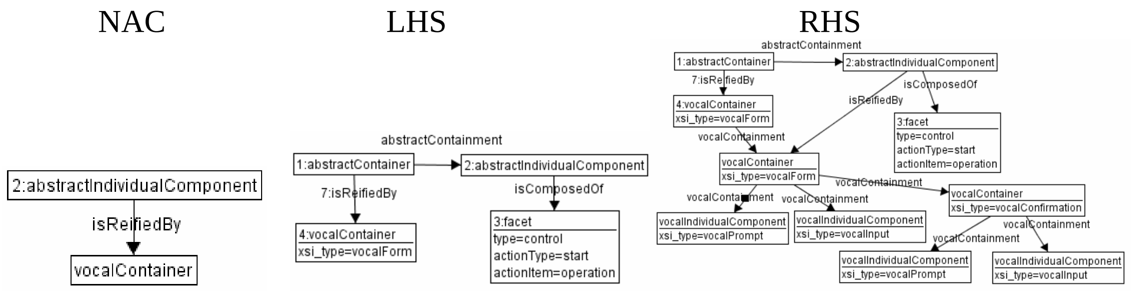


Figure 5-40. Generation of a vocal form that contains the dialog between the system and the user for the send questionnaire task

Sub-step 3.3: Arrangement of CICs

For each couple of adjacent AIOs that are reified into vocalCIOs, we define a vocalAdjencency relationship between these vocalCIOs that specify a delay time of 1 second. As vocalCIOs can be of two types (i.e., GCs or GICs), there are four possible combination to take into account. For each combination a specific rule is considered (Figure 5-41 and Figures B-13, B-14 and B-15 in Annex B).

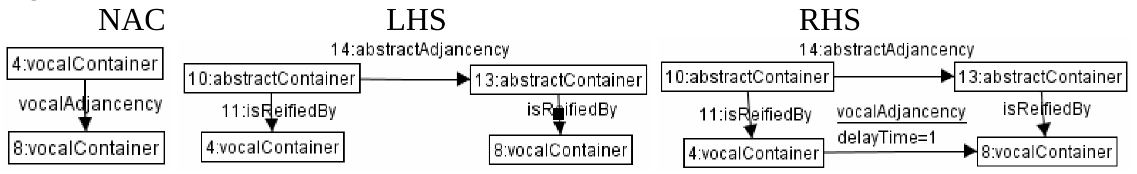


Figure 5-41. Generation of Vocal Adjacency relationships for <VC, VC> couples

Sub-step 3.4: Navigation definition

The rules that ensure the navigation definition are not used for this case study as all the components of the virtual polling system are presented into the same vocalGroup. For a complete set of rules defining the navigation definition based on design options, please refer to Section 5.3.

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 vocalCIOs that reify them is realized. As vocalCIOs are of two types (i.e., VCs and VICs), four rules describing the four possible combinations are considered (Figure 5-42 and Figures B-16, B-17 and B-18 in Annex B).

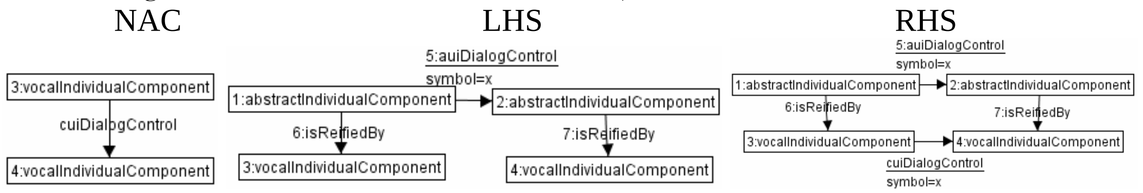

NAC
LHS
RHS
3:vocalIndividualComponent
cuiDialogControl
4:vocalIndividualComponent
5:auiDialogControl
symbol=x
1:abstractIndividualComponent
2:abstractIndividualComponent
6:isReifiedBy
7:isReifiedBy
3:vocalIndividualComponent
4:vocalIndividualComponent
5:auiDialogControl
symbol=x
1:abstractIndividualComponent
2:abstractIndividualComponent
6:isReifiedBy
7:isReifiedBy
3:vocalIndividualComponent
4:vocalIndividualComponent
cuiDialogControl
symbol=x

Figure 5-42. Generation of Concrete Dialog Control relationships for <VIC, VIC> couples

Sub-step 3.6: Derivation of CUI to Domain Relationship

Figure 5-43 illustrates the rule used to synchronize VICs with the corresponding attribute of an object from the Domain Model. The **updates** relationship is transposed from the AIC that is reified by the VIC.

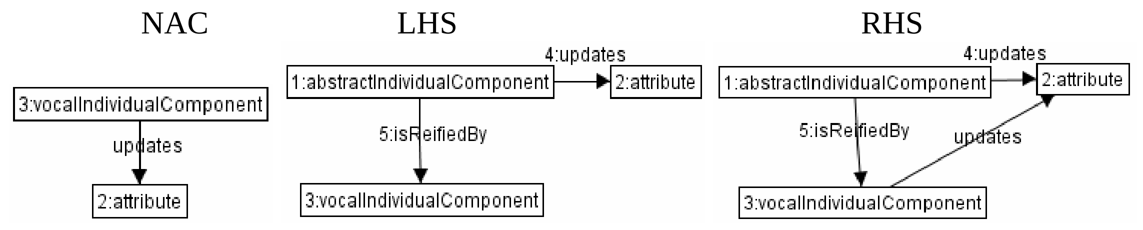
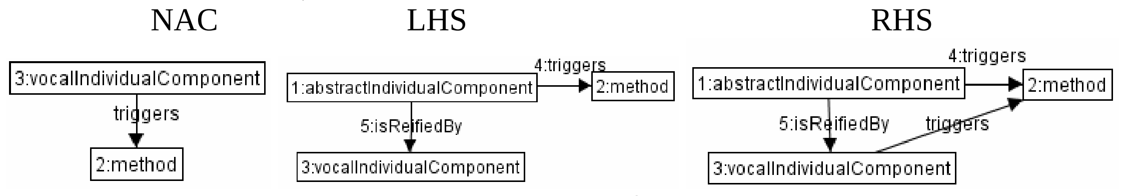

NAC
LHS
RHS
3:vocalIndividualComponent
updates
2:attribute
1:abstractIndividualComponent
4:updates
2:attribute
5:isReifiedBy
3:vocalIndividualComponent
1:abstractIndividualComponent
4:updates
2:attribute
5:isReifiedBy
updates
3:vocalIndividualComponent
Figure 5-43. Transposition of update relationship

Figure 5-44 illustrates the rule used to synchronize VICs with the corresponding method of an object from the Domain Model. The **triggers** relationship is transposed from the AIC that is reified by the VIC.


NAC
LHS
RHS
3:vocalIndividualComponent
triggers
2:method
1:abstractIndividualComponent
4:triggers
2:method
5:isReifiedBy
3:vocalIndividualComponent
1:abstractIndividualComponent
4:triggers
2:method
5:isReifiedBy
triggers
3:vocalIndividualComponent
Figure 5-44. Transposition of triggers relationship

Case 3: generation of multimodal UI – graphical predominance

This sub-case of our case study generates a multimodal UI where the graphical interaction is employed in a higher proportion then the vocal one. Thus, only the **Answer question** task will support a vocal interaction, the remaining tasks being accomplished using the graphical modality. In this section we will emphasize the reusability of our transformational rules. More precisely we will apply some rules designed for **Case 1: generation of graphical UI** and for **Case 2: generation of vocal UI,** to the graphical and respectively the vocal components that are part of the multimodal UI.

Sub-step 3.1: Reification of AC into CC

The rule described in Figure 5-45 creates a GC of type **window** that contains a **main box** of the application and a VC of type **vocalGroup.** Both containers reify the top most AC of the virtual polling system.

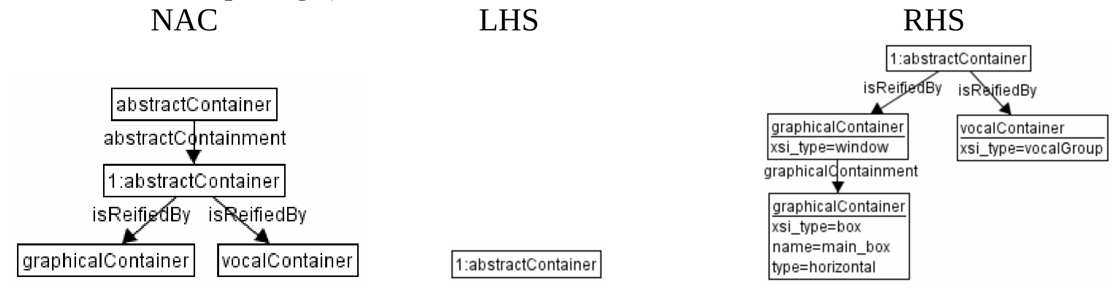


Figure 5-45. Creation of graphical and vocal top most containers

Figure 5-46 describes the rule that reifies each AC contained into the top most AC in a GC of type **box** and a VC of type **vocalForm**. The VC will be contained into the

vocalGroup created in the previous rule, while the GC will be embedded into the **main box** of the interface.

NAC LHS RHS

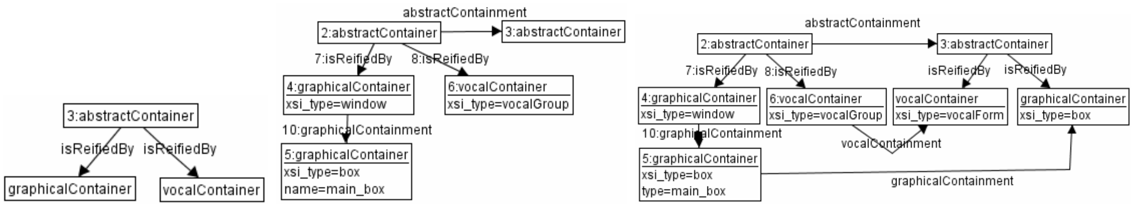


Figure 5-46. Creation of graphical and vocal containers embedded into the top most containers

Sub-step 3.2: Selection of CICs

In order to select the appropriated CICs, we apply the same set of rules described in Case 1 – Sub-step 3.2. The only modifications concerns the rules dedicated to the **Answer question** task (graphical interaction ensured by the rules in Figures 5-26 and 5-27 which are replaced by the rules illustrated in Figures 5-47 and 5-48 designed for multimodal interaction), respectively. The design options taken into consideration for this task are described in Table 5-1.

Design option	**Value**	**CIC**
Prompting	Graphical (assignment)	outputText
Grouping for input	Vocal (assignment)	vocalInput
Immediate feedback	Vocal (assignment)	vocalFeedback
Guidance for immediate feedback	Iconic (assignment)	imageComponents (speaker icon)

Table 5-1. Design option values for **Answer question** task

The rules that ensure the creation of the CICs according to the design option described above are detailed in the following. Figure 5-47 generates a VC of type **vocalMenu** (announcing the beginning of the questionnaire) that embeds two VICs: a **vocalInput** allowing users to input their answers and a **vocalFeedback** returning the recognized answer. A GIC of type **imageComponent** is also generated in order to guide the user with the feedback modality. Each **selectionValue** of an input facet of type select is reified by a VIC of type **vocalMenuItem** (Figure 5-48). Moreover, the **defaultContent** attribute of the **vocalMenuItem** takes the value of the **name** attribute of the corresponding selection value.

NAC LHS RHS

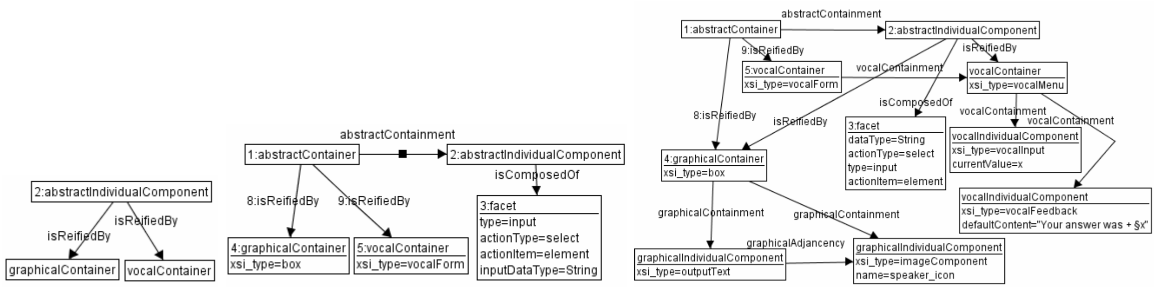


Figure 5-47. Creation of a vocalMenu that contains a vocalInput when an input facet of type select is encountered

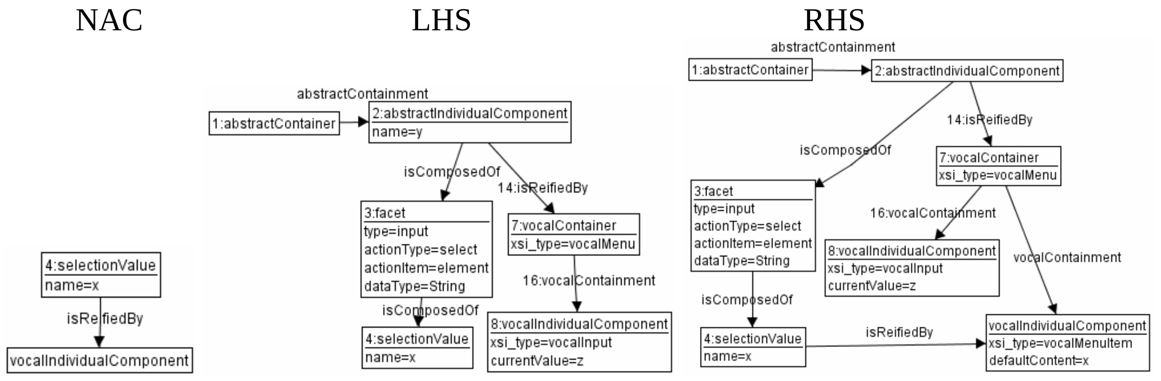


Figure 5-48. Generation of vocalMenuItems for each selection value of an input facet of type select

Sub-step 3.3: Synchronization of CICs

Due to the fact that the **outputText** component generated in the previous sub-step is not a GIC that allows gathering input from the user, no synchronization is necessary with the VIC of type **vocalInput** previously generated.

Sub-step 3.4: Arrangement of CICs

Rules illustrated in Figures 5-30, B-7, B-8 and B-9 are used to specify the arrangement of graphical CICs. For vocal adjacency relationship, Figures 5-41, B-13, B-14 and B-15 describe the rules used to perform the CICs arrangement.

Sub-step 3.5: Navigation definition

No navigation is defined as all the graphical components of the virtual polling system are presented into the same window and all the vocal components are embedded into the same vocalGroup.

Sub-step 3.6: Concrete Dialog Control Definition

For graphical components the dialog control is ensured by applying the rules described in Figures 5-31, B-10, B-11 and B-12, while Figures 5-42, B-16, B-17 and B-18 are illustrating the rules that define the control of vocal dialog.

Sub-step 3.7: Derivation of CUI to Domain Relationship

In order to synchronize the GICs with attributes and/or methods of classes from the Domain Model, the rules described in Figures 5-32 and 5-33 are reused. For VICs the reused rules are illustrated in Figures 5-43 and 5-44.

Case 4: generation of multimodal UI – vocal predominance

This sub-case generates a multimodal UI where the vocal interaction is employed in a higher proportion then the graphical one. Thus, the **Insert name** and **Send questionnaire** tasks are fulfilled using the graphical interaction, **Insert zip code** task can be accomplished employing the graphical or the vocal interaction, while the rest of the tasks (**Select gender**, **Select age category** and **Answer question**) are available only for vocal interaction.

Sub-step 3.1: Reification of AC into CC

The two rules illustrated in Case 3 - Sub-step 3.1 are reused for the reification of AC into CC (Figures 5-45, 5-46).

Sub-step 3.2: Selection of CICs

The rules designed in Figures 5-25 and 5-29 are reused in order to allow user to accomplish graphically the **Insert name** and **Send questionnaire** tasks, respectively. For the **Answer question** task, the rules illustrated in Figures 5-47 and 5-48 are executed.

Table 5-2 presents the design options and their CICs concretization for the **Insert zip code** task for which a multimodal **inputText** widget is generated.

Design option	Value	CIC
Prompting	Multimodal (redundancy)	outputText + vocalPrompt
Grouping for input	Multimodal (equivalence)	inputText + vocalInput
Immediate feedback	Graphical (assignment)	inputText
Guidance for input	Iconic (assignment)	imageComponents (microphone icon + keyboard icon)

Table 5-2. Design option values for multimodal **inputText** widget

Figure 5-49 we illustrate the rule applied in order to satisfy the design options described above. For the graphical part of the rule a GC of type **box** is created that embeds two GICs: an **inputText** and an **outputText**. Moreover, another two GICs of type **imageComponent** ensures the guidance concerning the available input modalities. For the vocal part a VC of type **vocalForm** is created and embeds two VICs: a **vocalPrompt** inviting the user to input the zip code and a **vocalInput** that will gather the users recognized answer into the **currentValue** attribute.

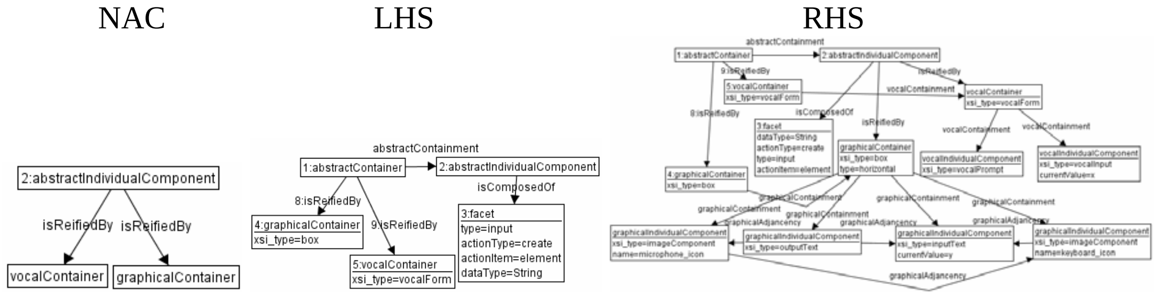


Figure 5-49. Creation of multimodal inputText with guidance

Table 5-3 presents the design options and their CICs concretization for the **Select gender** and **Select age category** tasks for which multimodal groups of radio buttons are generated.

Design option	Value	CIC
Prompting	Multimodal (redundancy)	outputText + vocalMenu
Grouping for input	Vocal (assignment)	vocalInput
Immediate feedback	Graphical (assignment)	radioButton
Guidance for input	Iconic (assignment)	imageComponents (microphone icon)

Table 5-3. Design option values for multimodal radioButtons

Figures 5-50 and 5-51 illustrate the rules applied in order to satisfy the design options described above. The first rule (Figure 5-50) creates a GC of type **box** that will embed a **group of multimodal radio buttons** when an input facet of type select element is encountered. For the graphical part of the rule, a GIC of type **outputText** representing the label associated to this group is created. The **defaultContent** of the GIC is the same as the name of the AIC. For the vocal part of the rule a VC of type **vocalMenu** will embed a VIC of type **vocalInput** that will contain in the **currentValue** attribute the result of the recognized input. The radio buttons associated to this group are created by executing the rule described in Figure 5-51. For each selection value of a facet of type select, two components will reify it:

- A GIC of type **radio button**, that has the same defaultContent as the name of the selection value
- A VIC of type VocalMenuItem, which has the same defaultContent as the name of the selection value.

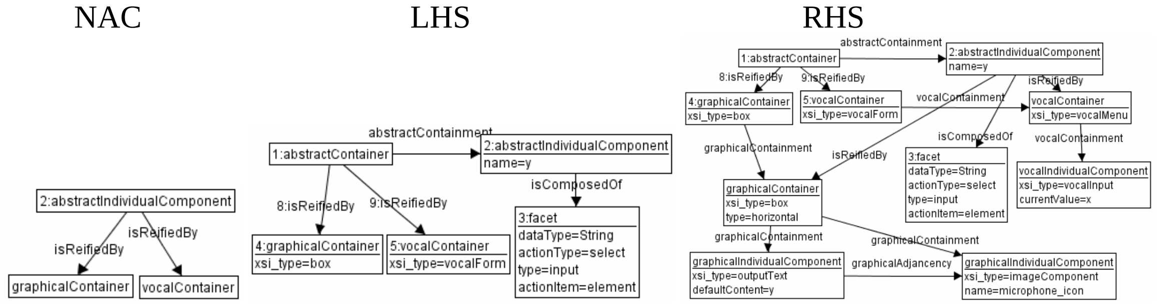


Figure 5-50. Generation of containers that will embed multimodal radio buttons

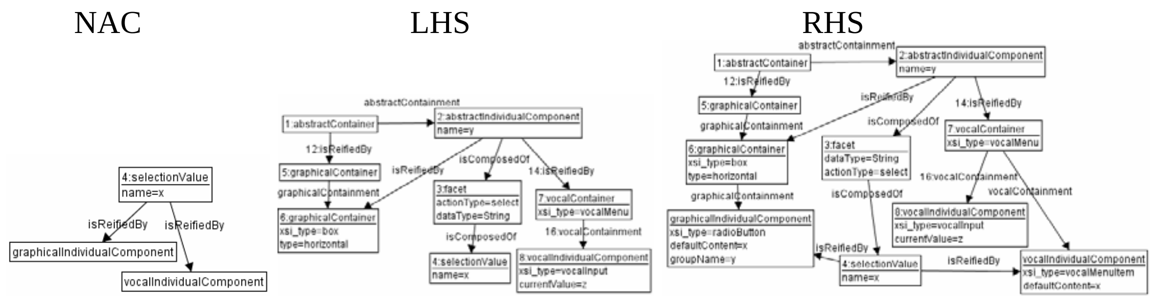


Figure 5-51. Generation of radioButtons and vocalMenuItems for each selection value of a facet of type select

Sub-step 3.3: Synchronization of CICs

Rule illustrated in Figure 5-52 creates the synchronization relationship between the VIC of type **vocalInput** and GIC of type **inputText** generated in Figure 5-49. The synchronization is defined between the currentValue **x** of the VIC and the currentValue **y** of the GIC.

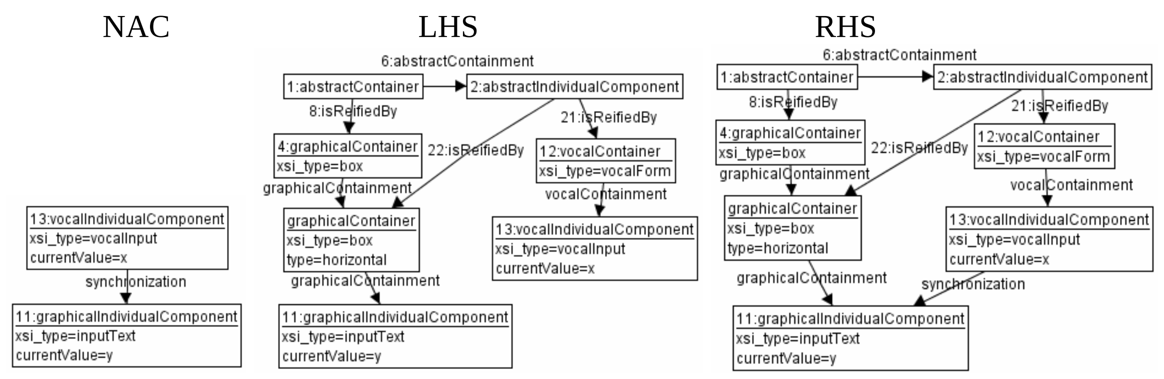


Figure 5-52. Synchronization between a vocalInput and an inputText

Rule illustrated in Figure 5-53 creates the synchronization relationship between the VIC of type **vocalInput** and the GC of type **box** generated in Figure 5-50 and embedding the group of radio buttons.

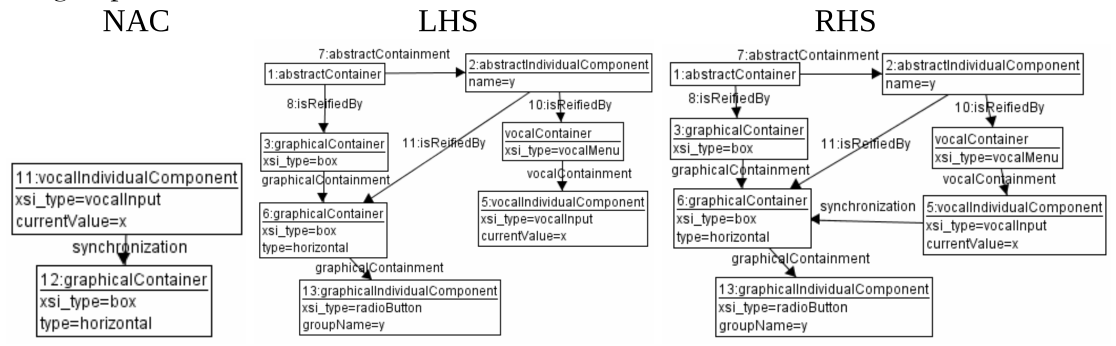


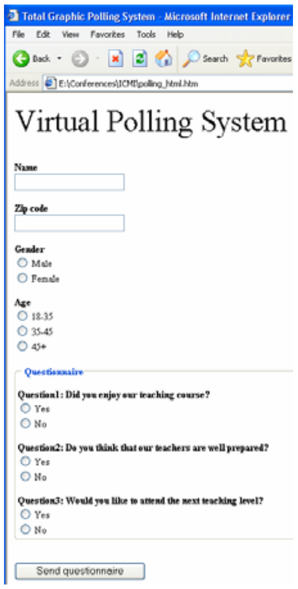
Figure 5-53. Synchronization between vocalInput and a box that embed a set of radio buttons

For the following sub-steps the same set of rules are reused as in their Case 3 counterpart:

Sub-step 3.4: Arrangement of CICs
Sub-step 3.5: Navigation definition
Sub-step 3.6: Concrete Dialog Control Definition
Sub-step 3.7: Derivation of CUI to Domain Relationship

5.2.4 Step 4: From CUI Model to FUI

This step consists of transforming each variant of the CUI into its respective FUI specification. In the following we illustrate the result of the interpretation of the FUIs with their corresponding browsers. Thus, Figure 5-54 show the resultant graphical UI interpreted with Internet Explorer browser, while Figures 5-55 and 5-56 present the multimodal UI interpreted with NetFront multimodal browser for graphical predominant and vocal predominant UI, respectively. Figure 5-57 is a textual representation of the vocal UI (C= Computer, U= User).



Total Graphic Polling System - Microsoft Internet Explorer

Address E:\Conferences\ICMI\polling_html.htm

Virtual Polling System

Name

Zip code

Gender
Male
Female

Age
18-35
35-45
45+

Questionnaire

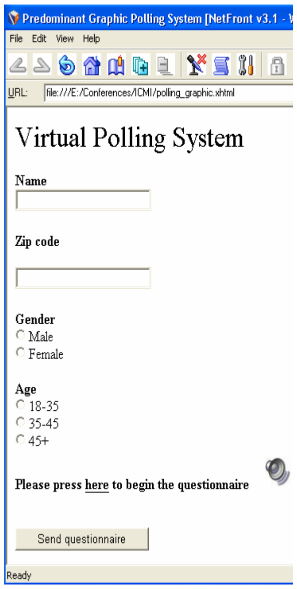
Question1 : Did you enjoy our teaching course?
Yes
No

Question2: Do you think that our teachers are well prepared?
Yes
No

Question3: Would you like to attend the next teaching level?
Yes
No

Send questionnaire

Fig. 5-54. Graphical UI



Predominant Graphic Polling System [NetFront v3.1 -

URL: file:///E:/Conferences/ICMI/polling_graphic.xhtml

Virtual Polling System

Name

Zip code

Gender
Male
Female

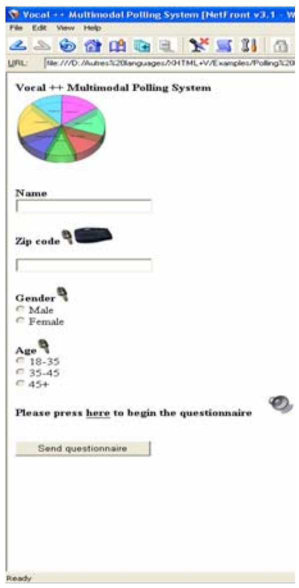
Age
18-35
35-45
45+

Please press here to begin the questionnaire

Send questionnaire

Ready

Fig. 5-55. Multimodal UI – graphical predominance



Vocal ++ Multimodal Polling System [NetFront v3.1 -

URL: file:///D:/Autres%20languages/XHTML+V/Examples/Polling%20

Vocal ++ Multimodal Polling System

Name

Zip code

Gender
Male
Female

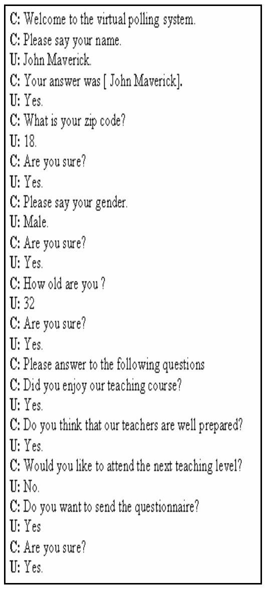
Age
18-35
35-45
45+

Please press here to begin the questionnaire

Send questionnaire

Ready

Fig. 5-56. Multimodal UI – vocal predominance



C: Welcome to the virtual polling system.
C: Please say your name.
U: John Maverick.
C: Your answer was [John Maverick].
U: Yes.
C: What is your zip code?
U: 18.
C: Are you sure?
U: Yes.
C: Please say your gender.
U: Male.
C: Are you sure?
U: Yes.
C: How old are you ?
U: 32
C: Are you sure?
U: Yes.
C: Please answer to the following questions
C: Did you enjoy our teaching course?
U: Yes.
C: Do you think that our teachers are well prepared?
U: Yes.
C: Would you like to attend the next teaching level?
U: No.
C: Do you want to send the questionnaire?
U: Yes
C: Are you sure?
U: Yes.

Fig. 5-57. Vocal UI

5.3 Case study 2: Car Rental System

The second case study is dedicated to an on-line car rental system that allows user to select, to search and to pay for a car to rent depending of their preferences. Comparing to the Virtual polling system, the present case study aims at producing only graphical and multimodal UIs while still emphasizing their corresponding development design options presented in Chapter 4.2. A total vocal UI might also be taken into account, but it is out of the purpose of this section.

The scenario is as follows (Figure 5-58):

- Describe the *Task* and *Domain Models*

- Generate two AUIs based on the two different values of the sub-task presentation design option (i.e., separated windows and combined group boxes)
- For each AUI two CUIs are derived, a graphical one and a multimodal one
- Four FUIs are derived corresponding to each CUI obtained in the previous step.

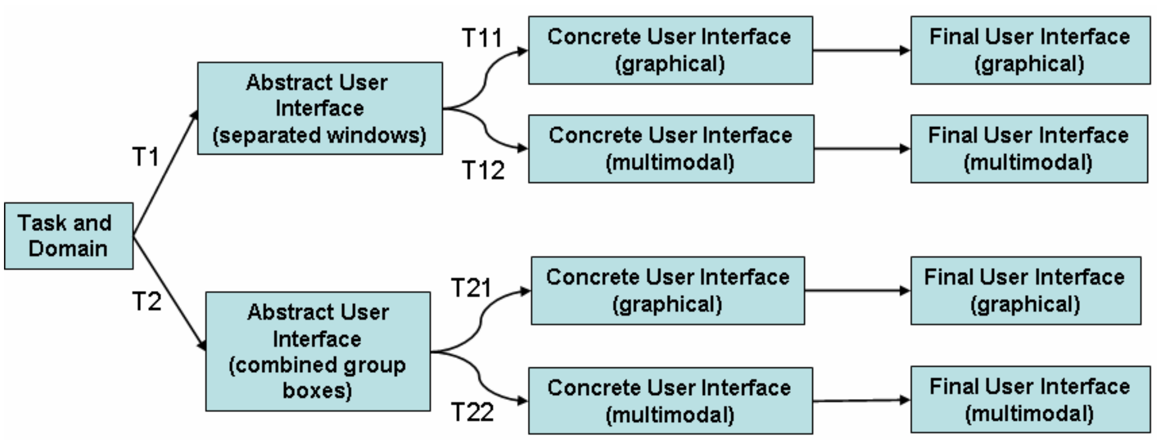

Task and Domain
T1
T2
Abstract User Interface (separated windows)
Abstract User Interface (combined group boxes)
T11
T12
T21
T22
Concrete User Interface (graphical)
Concrete User Interface (multimodal)
Final User Interface (graphical)
Final User Interface (multimodal)

Figure 5-58. Development scenario for car rental system

5.3.1 Step 1: The Task and Domain Models

The root task of the Task Model (Figure 5-59) is decomposed into three basic sub-tasks:

1. **Determine rental preferences** (Figure 5-60): 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 5-61): 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 5-62): 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.

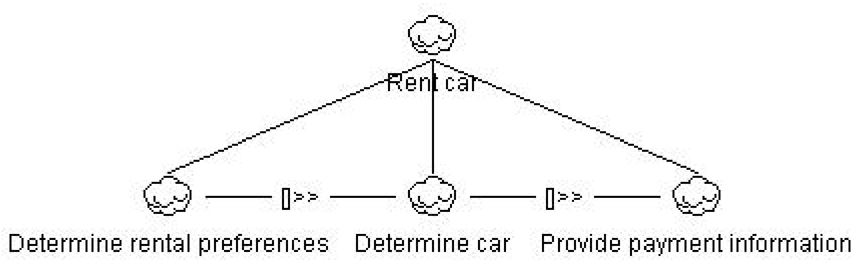


Figure 5-59. Root task decomposition

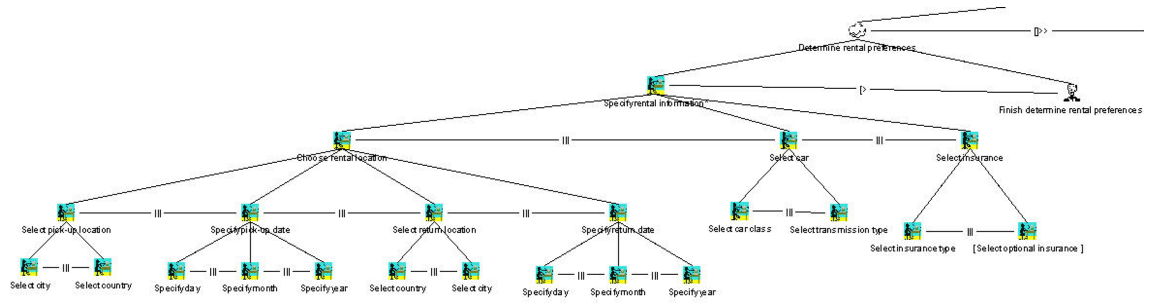


Figure 5-60. The decomposition of Determine rental preferences sub-task

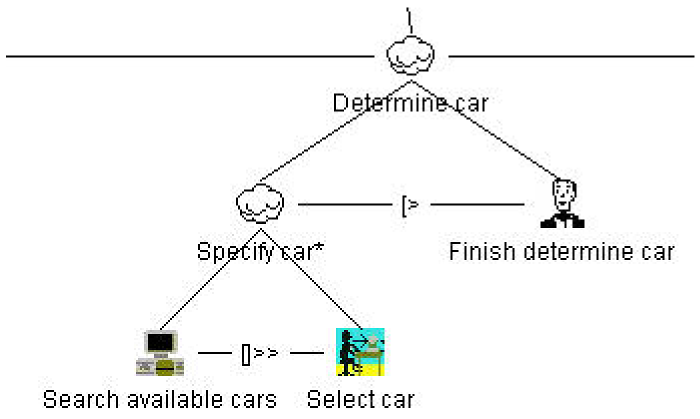


Figure 5-61. The decomposition of Determine car sub-task

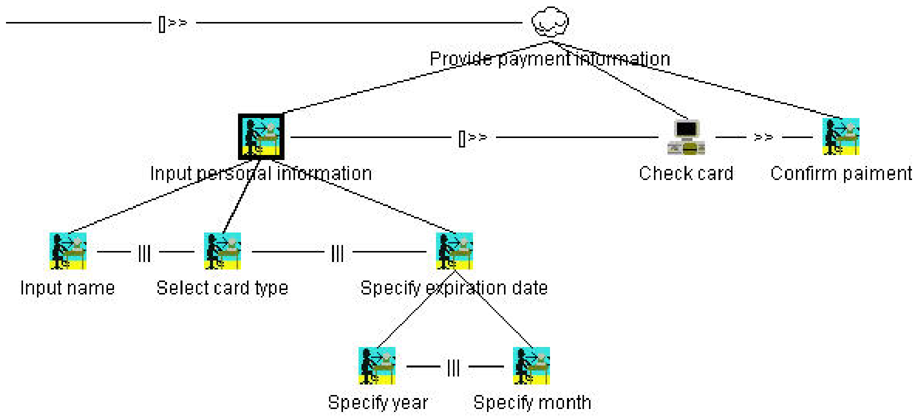
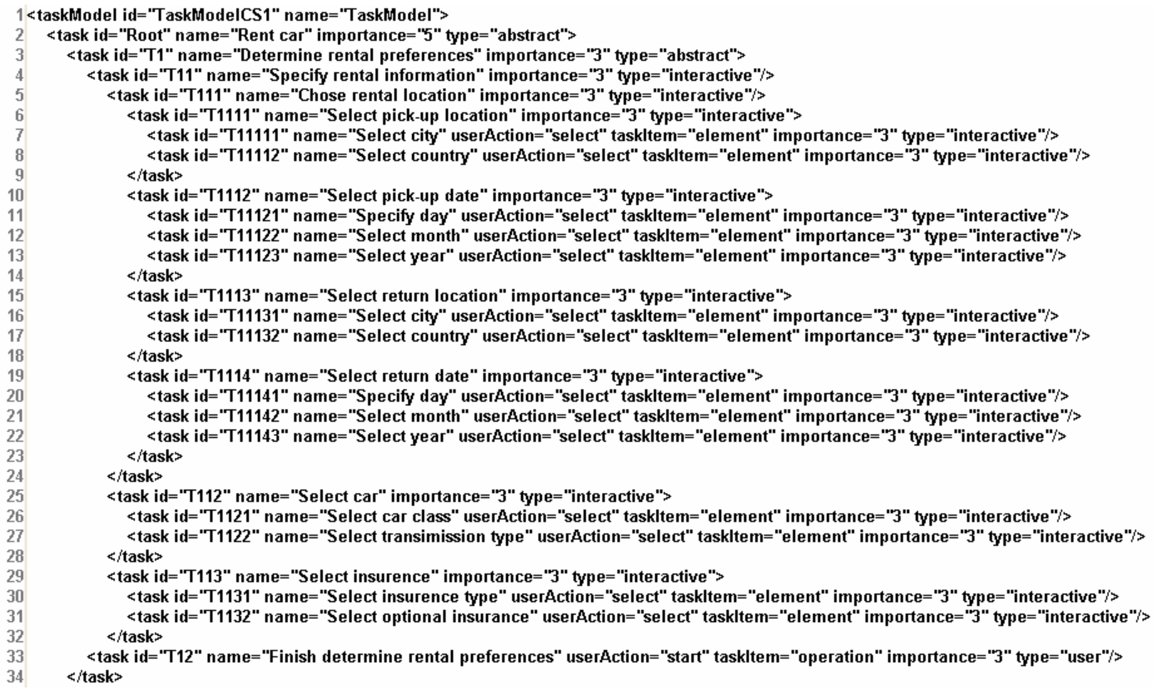


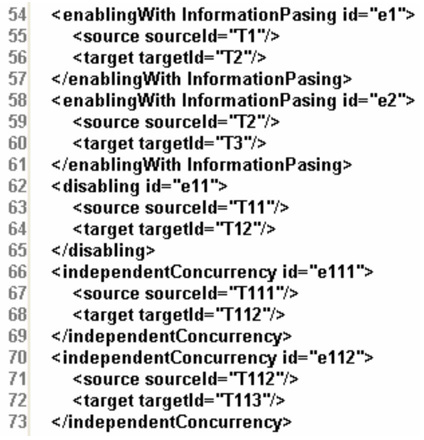
Figure 5-62. The decomposition of Provide payment information sub-task

The UsiXML specification corresponding to the *Task Model* is generated by IdealXML. Figure 5-63 shows two excerpts: lines 1 to 34 illustrate the hierarchical decomposition of the tasks, while lines 54 to 73 describe the relationships between the tasks.



```xml
<taskModel id="TaskModelCS1" name="TaskModel">
  <task id="Root" name="Rent car" importance="5" type="abstract">
    <task id="T1" name="Determine rental preferences" importance="3" type="abstract">
      <task id="T11" name="Specify rental information" importance="3" type="interactive"/>
        <task id="T111" name="Chose rental location" importance="3" type="interactive"/>
          <task id="T1111" name="Select pick-up location" importance="3" type="interactive">
            <task id="T11111" name="Select city" userAction="select" taskItem="element" importance="3" type="interactive"/>
            <task id="T11112" name="Select country" userAction="select" taskItem="element" importance="3" type="interactive"/>
          </task>
          <task id="T1112" name="Select pick-up date" importance="3" type="interactive">
            <task id="T11121" name="Specify day" userAction="select" taskItem="element" importance="3" type="interactive"/>
            <task id="T11122" name="Select month" userAction="select" taskItem="element" importance="3" type="interactive"/>
            <task id="T11123" name="Select year" userAction="select" taskItem="element" importance="3" type="interactive"/>
          </task>
          <task id="T1113" name="Select return location" importance="3" type="interactive">
            <task id="T11131" name="Select city" userAction="select" taskItem="element" importance="3" type="interactive"/>
            <task id="T11132" name="Select country" userAction="select" taskItem="element" importance="3" type="interactive"/>
          </task>
          <task id="T1114" name="Select return date" importance="3" type="interactive">
            <task id="T11141" name="Specify day" userAction="select" taskItem="element" importance="3" type="interactive"/>
            <task id="T11142" name="Select month" userAction="select" taskItem="element" importance="3" type="interactive"/>
            <task id="T11143" name="Select year" userAction="select" taskItem="element" importance="3" type="interactive"/>
          </task>
        </task>
        <task id="T112" name="Select car" importance="3" type="interactive">
          <task id="T1121" name="Select car class" userAction="select" taskItem="element" importance="3" type="interactive"/>
          <task id="T1122" name="Select transimission type" userAction="select" taskItem="element" importance="3" type="interactive"/>
        </task>
        <task id="T113" name="Select insurence" importance="3" type="interactive">
          <task id="T1131" name="Select insurence type" userAction="select" taskItem="element" importance="3" type="interactive"/>
          <task id="T1132" name="Select optional insurance" userAction="select" taskItem="element" importance="3" type="interactive"/>
        </task>
      <task id="T12" name="Finish determine rental preferences" userAction="start" taskItem="operation" importance="3" type="user"/>
    </task>
```

....



```xml
  <enablingWith InformationPasing id="e1">
    <source sourceId="T1"/>
    <target targetId="T2"/>
  </enablingWith InformationPasing>
  <enablingWith InformationPasing id="e2">
    <source sourceId="T2"/>
    <target targetId="T3"/>
  </enablingWith InformationPasing>
  <disabling id="e11">
    <source sourceId="T11"/>
    <target targetId="T12"/>
  </disabling>
  <independentConcurrency id="e111">
    <source sourceId="T111"/>
    <target targetId="T112"/>
  </independentConcurrency>
  <independentConcurrency id="e112">
    <source sourceId="T112"/>
    <target targetId="T113"/>
  </independentConcurrency>
```

....

Figure 5-63. Excerpts of Task Model expressed in UsiXML

The *Domain Model* (Figure 5-64) 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 single payment mean considered in our system. **Coordinates** is a class used as data type by **RentalInformation** and **Client** classes.

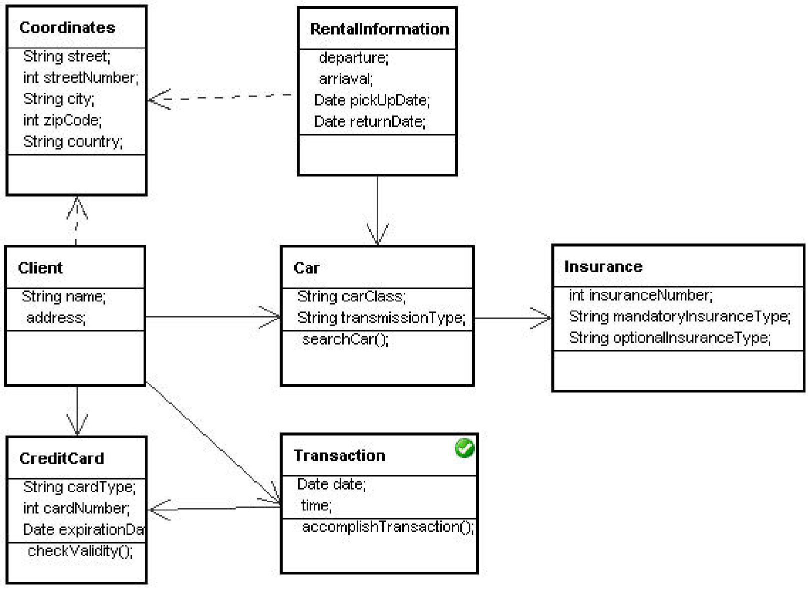
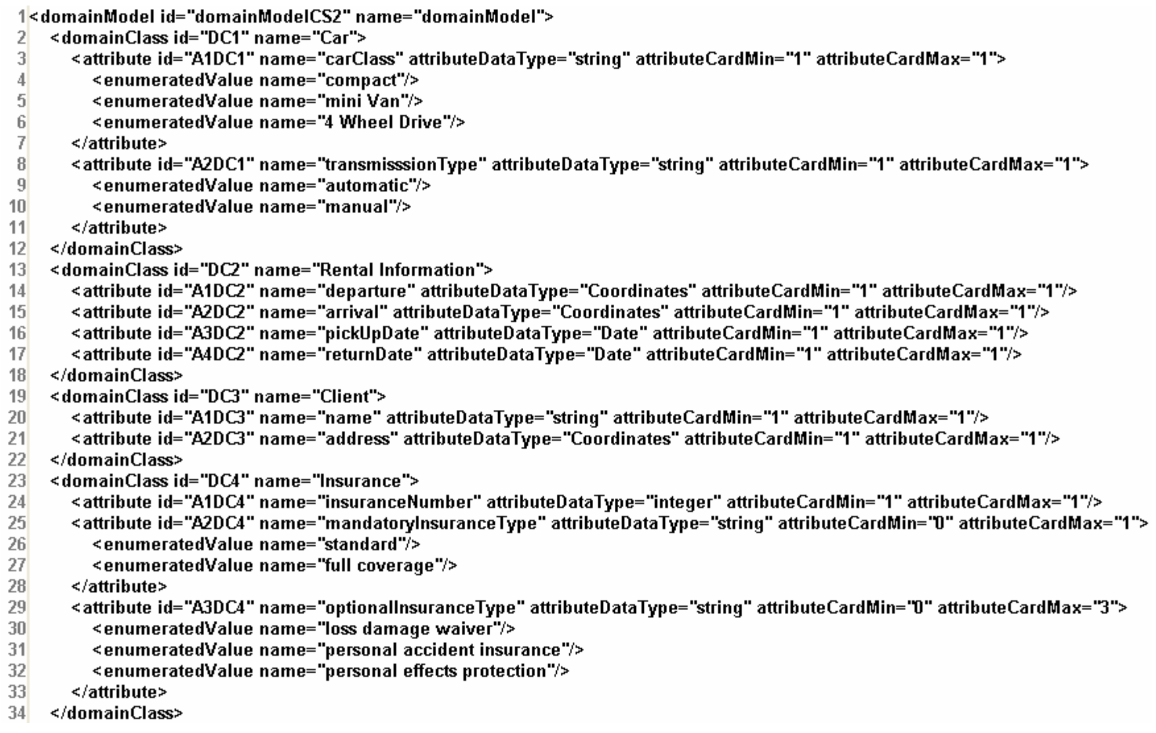


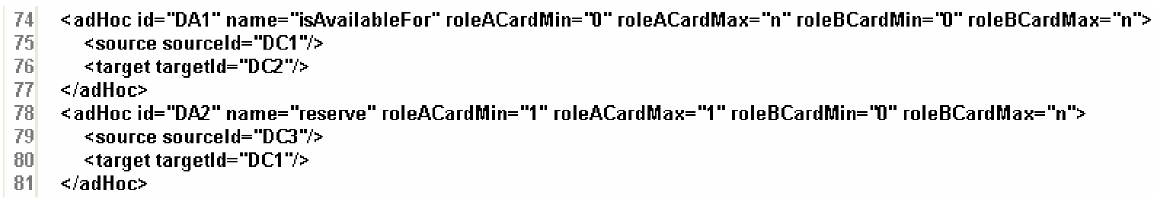
Figure5-64. Domain model for the car rental system

Figure 5-65 illustrate two excerpts of the *Domain Model* expressed in UsiXML language. Lines 1 to 34 define a part of the classes that are involved into the class diagram, while lines 74 to 81 describe the **adHoc** relationship between **Car** class and **RentalInformation** class and between **Client** class and **Car** class, respectively.



```
 1<domainModel id="domainModelCS2" name="domainModel">
 2   <domainClass id="DC1" name="Car">
 3      <attribute id="A1DC1" name="carClass" attributeDataType="string" attributeCardMin="1" attributeCardMax="1">
 4         <enumeratedValue name="compact"/>
 5         <enumeratedValue name="mini Van"/>
 6         <enumeratedValue name="4 Wheel Drive"/>
 7      </attribute>
 8      <attribute id="A2DC1" name="transmisssionType" attributeDataType="string" attributeCardMin="1" attributeCardMax="1">
 9         <enumeratedValue name="automatic"/>
10         <enumeratedValue name="manual"/>
11      </attribute>
12   </domainClass>
13   <domainClass id="DC2" name="Rental Information">
14      <attribute id="A1DC2" name="departure" attributeDataType="Coordinates" attributeCardMin="1" attributeCardMax="1"/>
15      <attribute id="A2DC2" name="arrival" attributeDataType="Coordinates" attributeCardMin="1" attributeCardMax="1"/>
16      <attribute id="A3DC2" name="pickUpDate" attributeDataType="Date" attributeCardMin="1" attributeCardMax="1"/>
17      <attribute id="A4DC2" name="returnDate" attributeDataType="Date" attributeCardMin="1" attributeCardMax="1"/>
18   </domainClass>
19   <domainClass id="DC3" name="Client">
20      <attribute id="A1DC3" name="name" attributeDataType="string" attributeCardMin="1" attributeCardMax="1"/>
21      <attribute id="A2DC3" name="address" attributeDataType="Coordinates" attributeCardMin="1" attributeCardMax="1"/>
22   </domainClass>
23   <domainClass id="DC4" name="Insurance">
24      <attribute id="A1DC4" name="insuranceNumber" attributeDataType="integer" attributeCardMin="1" attributeCardMax="1"/>
25      <attribute id="A2DC4" name="mandatoryInsuranceType" attributeDataType="string" attributeCardMin="0" attributeCardMax="1">
26         <enumeratedValue name="standard"/>
27         <enumeratedValue name="full coverage"/>
28      </attribute>
29      <attribute id="A3DC4" name="optionalInsuranceType" attributeDataType="string" attributeCardMin="0" attributeCardMax="3">
30         <enumeratedValue name="loss damage waiver"/>
31         <enumeratedValue name="personal accident insurance"/>
32         <enumeratedValue name="personal effects protection"/>
33      </attribute>
34   </domainClass>
```

……………………………………………………………………………………………………



```
<adHoc id="DA1" name="isAvailableFor" roleACardMin="0" roleACardMax="n" roleBCardMin="0" roleBCardMax="n">
   <source sourceId="DC1"/>
   <target targetId="DC2"/>
</adHoc>
<adHoc id="DA2" name="reserve" roleACardMin="1" roleACardMax="1" roleBCardMin="0" roleBCardMax="n">
   <source sourceId="DC3"/>
   <target targetId="DC1"/>
</adHoc>
```

Figure5-65. Excerpts of Domain Model expressed in UsiXML

The mappings between the Task Model and the Domain Model are summed-up in Table 5-4.

Task model		**Domain model**
Select pick-up country	(select element)	RentalInformation.departure.country
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 country	(select element)	RentalInformation.arrival.country
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 5-4. Mappings between task and domain models for the car rental system

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

Case 1: Presentation of the sub-tasks into separated windows

In this first case (T1 in Figure 5-58) we illustrate graph transformation rules applied on UsiXML models in order to generate a UI where the presentation of the sub-tasks is separated in three windows. The navigation between the windows is of type *sequential* and is concretized in a *global placement* of the (NEXT, PREV) buttons identified on each window. The navigation is ensured only by these two logically grouped objects, so the value of the cardinality is *simple*.

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

We present in this sub-step a series of rules that are designed to be suitable for the present case study. The rest of the rules (i.e., the rules that reconstruct containment relationship between ACs and between ACs and AICs) are the same used in the first case study (Figures 5-10 and 5-11). In order to generate an AC for each sub-task of the root task, rule illustrated in Figure 5-66 is applied. For the each leaf task a corresponding AIC will be created by applying the rule from Figure 5-8.

NAC LHS RHS

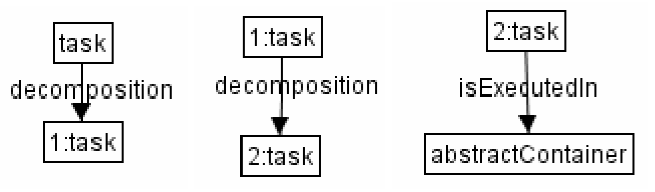


Figure 5-66. Generate abstract containers for each sub-task of the root task

The rule presented in Figure 5-67 creates an AC that embeds two AICs which will reify later the two buttons (Previous, Next) that will ensure the navigation between the ACs. The rule is applied for each task that disables iterative tasks.

NAC LHS RHS

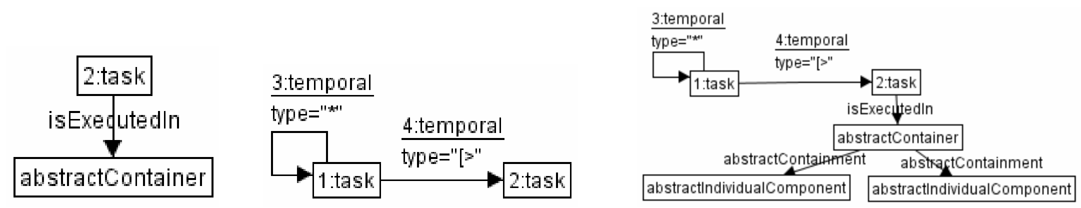


Figure 5-67. Create two abstract individual components for task that disable iterative tasks

Sub-step 2.2: Rules for the selection of the AICs

Based on the mappings that exist between the Task Model and the Domain Model different facets will be created for the AICs that support the execution of the tasks. In the following we present the rules applied to create these facets:

- Input facet of type select element for the AICs (Figures 5-13) assigned to the following tasks: **select country**, **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**); 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 (Figure 5-14)
- Input facet of type create element for the AICs assigned to the **input name** and **input card number** tasks (Figure 5-12)
- Navigation facet of type start operation for the AICs that ensure the navigation between the ACs (Figure 5-68)

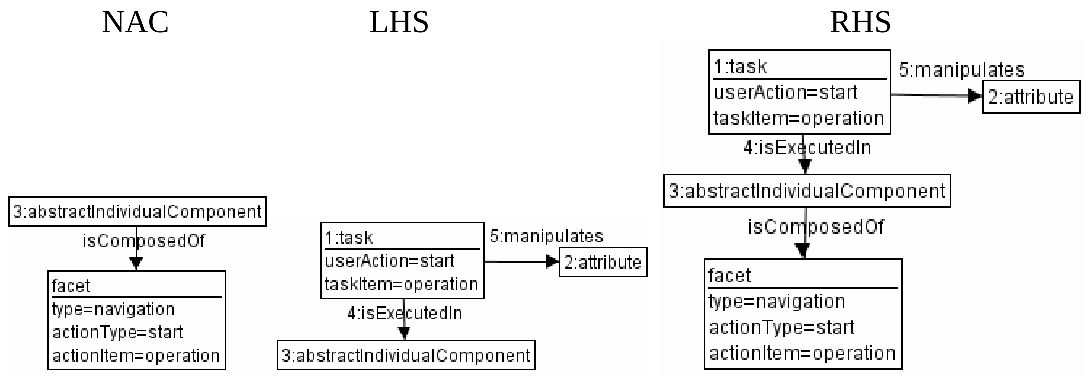


Figure 5-68. Create navigation facet for AIC executed in tasks of type start operation

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

The same set of rules identified in Section 5.2, Sub-step 2.3 will be reused in order to create abstractAdjancency relationships between AIOs used to execute couple of sister tasks (Figures 5-17, B-1, B-2 and B-3).

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

For each couple of sister tasks executed into AIOs, an abstractDialogControl relationship is defined between these AIOs that have the same semantics as the temporal relationship defined between the tasks. In order to define the abstract dialog control we are reusing the rules illustrated in Section 5.2, Sub-step 2.4 (Figures 5-18, B-4, B-5 and B-6).

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

Rules described in Section 5.2, Sub-step 2.5 are used to define mapping relationships of type updates and triggers that allow synchronizing AICs with attributes (Figure 5-19) and methods (Figure 5-20) from the *Domain Model.*

Case 2: presentation of the sub-tasks into combined group boxes

In the second case (T2 in Figure 5-58) we illustrate graph transformation rules applied on UsiXML models in order to generate a UI where the presentation of the sub-tasks is combined in three group boxes displayed in the same window. The navigation between the group boxes is of type *sequential* and is concretized in a *global placement* of the (OK, CANCEL) buttons. The navigation is ensured only by these two logically grouped objects, so the value of the cardinality is *simple.*

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

The generation of the top-most AC is illustrated in Figure 5-69. This AC embeds the two AICs that will be reified later into the two buttons which will ensure the navigation (Figure 5-70). For each sub-task of the root task an AC will be generated (Figure 5-71). These ACs are an abstraction of the groupBoxes that will be further reified from it. In order to generate the AICs corresponding to each leaf task, rule illustrated in Figure 5-8 is applied. The rules that reconstruct containment relationship between ACs and between ACs and AICs are the same used in the first case study (Figures 5-10 and 5-11).

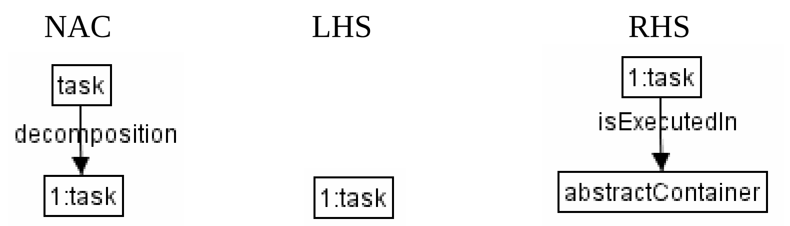


Figure 5-69. Generate an abstract container for the root task

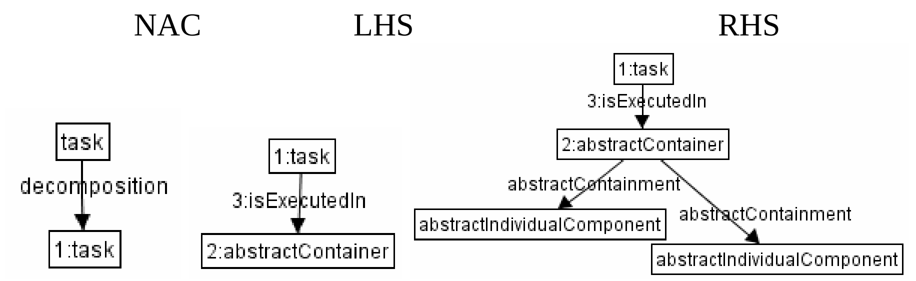


Figure 5-70. Generate two abstract individual components implicated later in the navigation

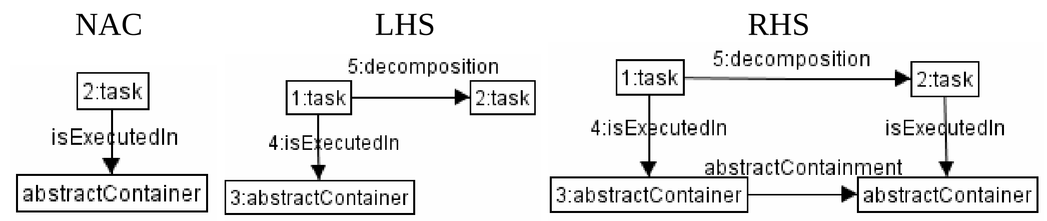


Figure 5-71. Generate abstract containers for each sub-task of the root task

Sub-step 2.2: Rules for the selection of the AICs

The same set of rules identified in Case 1 – Sub-step 2.2 will be applied.

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

The same set of rules identified in Section 5.2, Sub-step 2.3 will be reused in order to create abstractAdjancency relationships between AIOs used to execute couple of sister tasks (Figures 5-17, B-1, B-2 and B-3).

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

For each couple of sister tasks executed into AIOs, an abstractDialogControl relationship if defined between these AIOs that have the same semantics as the temporal relationship between the tasks. In order to define the abstract dialog control we are reusing the rules illustrated in Section 5.2, Sub-step 2.4 (Figures 5-18, B-4, B-5 and B-6).

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

Rules described in Section 5.2, Sub-step 2.5 is reused to define mapping relationships of type *updates* and *triggers* that allow synchronizing AICs with attributes (Figure 5-19) and methods (Figure 5-20) from the *Domain Model.*

5.3.3 Step 3: From AUI Model to CUI Model

For each AUI obtained in the previous step (i.e., Case 1: Presentation of the sub-tasks into separated windows and Case 2: Presentation of the sub-tasks into combined group boxes), two sub-cases will be taken into consideration. Thus, four CUIs will be derived in the current step:

- Sub-case 1.1: graphical CUI presented into separated windows
- Sub-case 1.2: multimodal CUI presented into separated windows
- Sub-case 2.1: graphical CUI presented into combined group boxes
- Sub-case 2.2: multimodal CUI presented into combined group boxes

Sub-case 1.1: graphical CUI presented into separated windows

The present sub-case (T11 in Figure 5-58) contains transformation rules applied on the AUI (separated window case) produced in the previous step, in order to generate the correspondent graphical CUI.

Sub-step 3.1: reification of AC into CC

Rule illustrated in Figure 5-72 creates two GCs, one of type **window** and one of type **box** for each of the three ACs generated in the previous step. In order to create the group boxes that will embed later the GICs, rule illustrated in Figure 5-24 is modified so as the GC of type **box** becomes a GC of type **groupBox**.

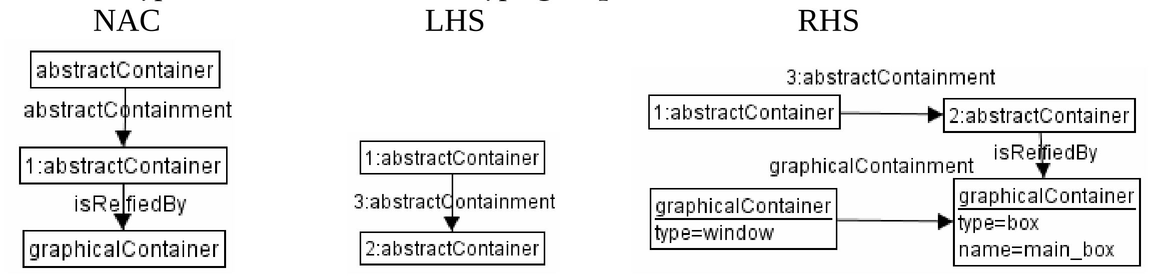


Figure 5-72. Create a window and the corresponding main box for each top-level AC

Sub-step 3.2: Selection of CICs

For the selection of CICs, a series of rules are applied. Table 5-5 shows the rule(s) applied in order to create GICs that will allow users to fulfill the corresponding task.

Task	Rule	Rule description
Input name Input card number	Figure 5-25	When an **input facet** of type **create element** is encountered, generate a GC of type **box** that will embed an **outputText** (the label) and an **inputText** (the textfield).
Select pick-up and return information (country, city, day, month, year) Select card type Select expiration date (month, year)	Figure 5-73	When an **input facet** of type select element is found, create a GC of type **box** that will embed two GICs: an **outputText** (the label) and a **comboBox**.
	Figure 5-74	For each selection value of an **input facet** of type **select element**, create an **item** for the previously generated comboBox for which the value of the **defaultContent** attribute will be the same as the name of the **selection value**.

Select car	Figure 5-75	When an **input facet** of type **select element** is found, create a GC of type **box** that will embed two GICs: a **listBox** and an **outputText** (the associated label).
	Figure 5-76	For each selection value of an **input facet** of type **select element**, create an **item** for the previously generated **listBox** for which the value of the **defaultConten**t attribute will be the same as the name of the **selection value**.
Select optional insurance	Figure 5-77	When an **input facet** of type **select element** is found, create a GC of type **box** that will embed a GIC of type **outputText** representing the label associated to the future group of check boxes.
	Figure 5-78	For each selection value of an **input facet** of type **select element**, create a GIC of type **checkbox** for which the value of the **defaultContent** attribute will be the same as the name of the **selection value**.
Select car class Select transmission type	Figure 5-26 Figure 5-27	Generation of a group of radio buttons (See description in Section 5.2).
Go next/previous	Figure 5-79	When two **navigation facets** of type **start operation** are encountered, two GICs of type **button** (corresponding to the Previous and Next buttons) are generated.
Confirm payment	Figure 5-29	Generation of a **control button**. (See description in Section 5.2).

Table 5-5. Correspondance between task and rules

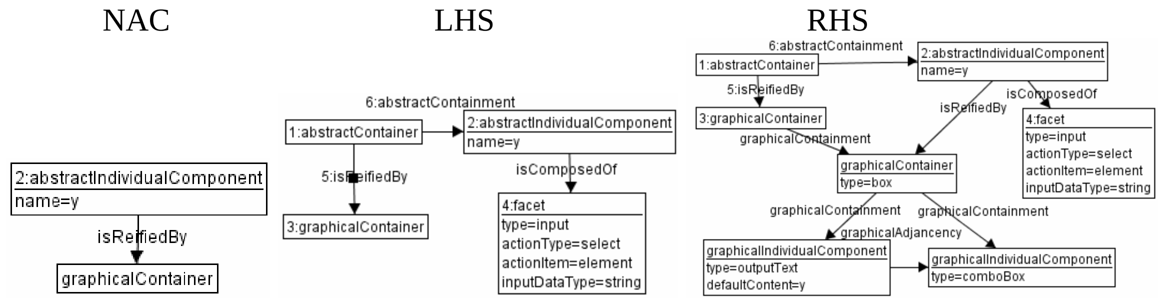


Figure 5-73. Generation of a GC of type box that contains an outputText and a comboBox

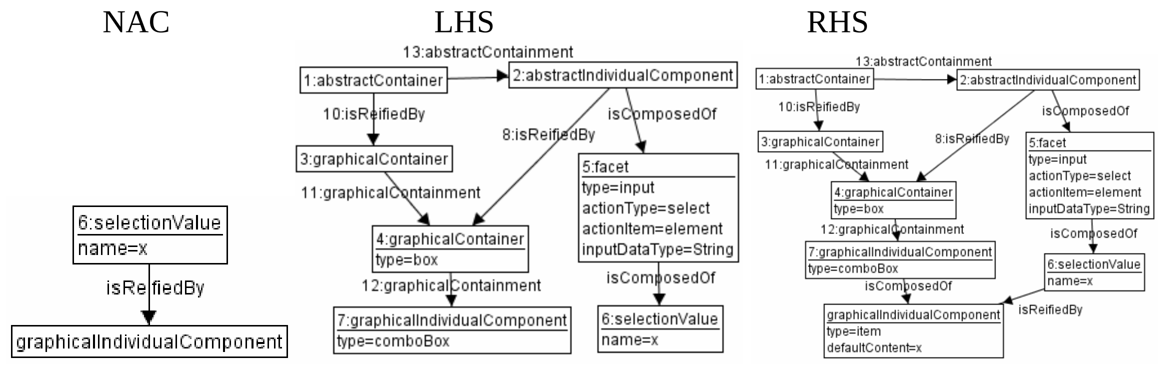


Figure 5-74. Generation of comboBox items for each selection value of a facet of type select

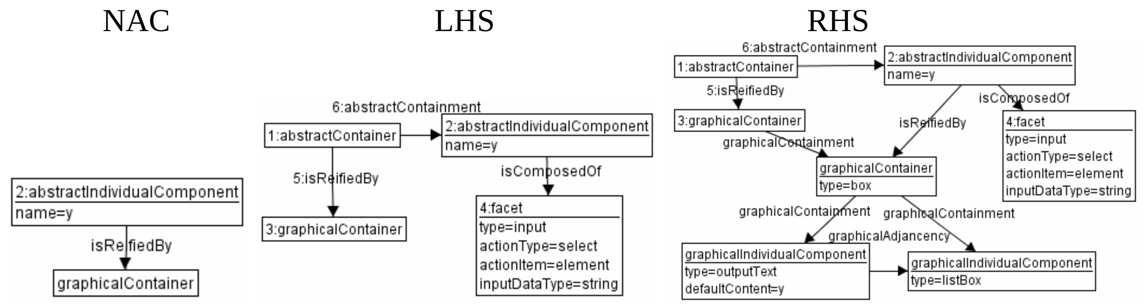


Figure 5-75. Generation of a GC of type box that contains an outputText and a listBox

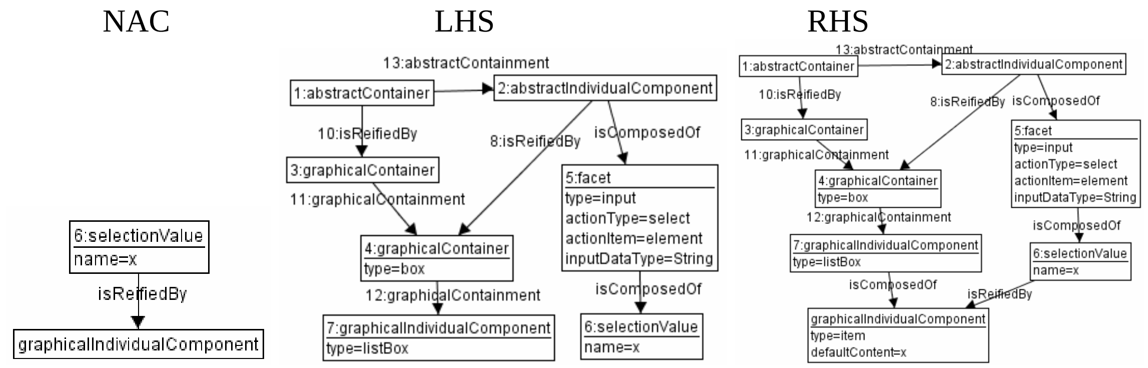


Figure 5-76. Generation of listBox items for each selection value of a facet of type select

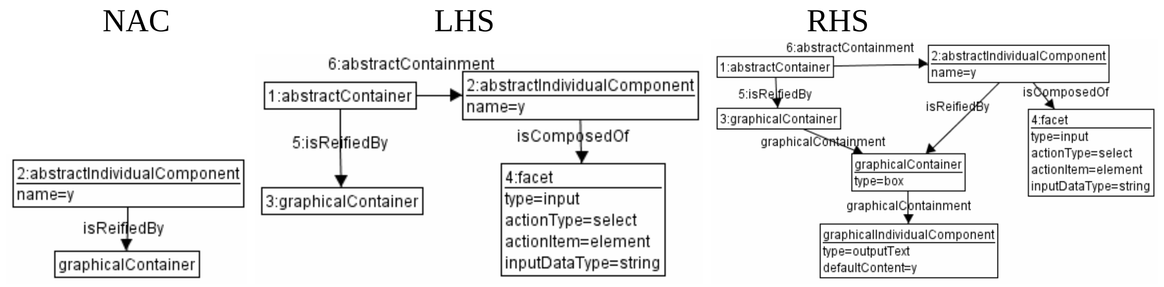


Figure 5-77. Generation of a GC of type box that contains an outputText

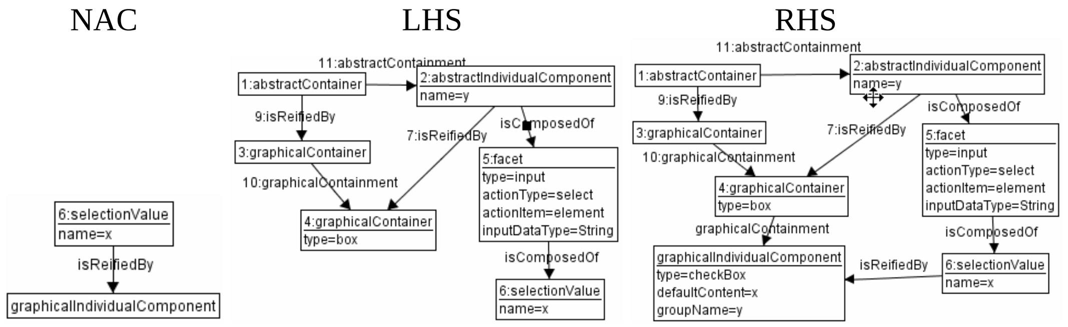


Figure 5-78. Generation of checkBoxes for each selection value of a facet of type select

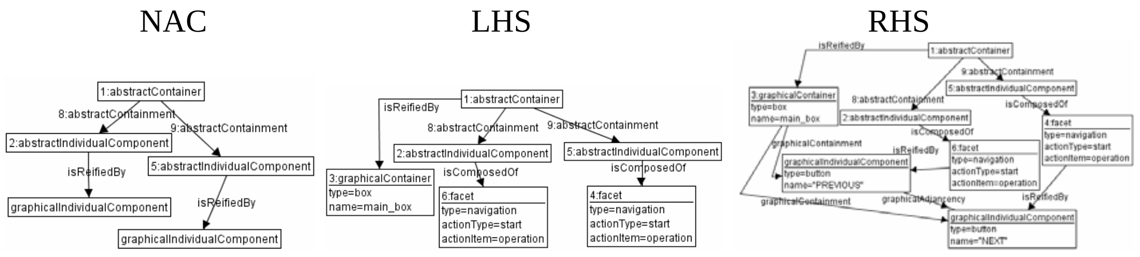


Figure 5-79. Generation of navigation buttons

Sub-step 3.3: Arrangement of CICs

Rules presented in Figures 5-30, B-7, B-8, B-9 are reused in order to accomplish this sub-step.

Sub-step 3.4: Navigation definition

For the definition of the navigation we consider the **sub-task navigation** graphical design option presented in Section 4.2.1. As our sub-tasks are presented into three separated windows we have made the choice of a **sequential navigation** ensured by the **Previous and Next buttons** generated in Sub-step 3.2. Figures 5-80 and 5-81 generate the graphical relationships of type **graphicalTransition** that endows the **Previous and Next buttons** with **activation/deactivation** power over the adjacent and current GCs, respectively.

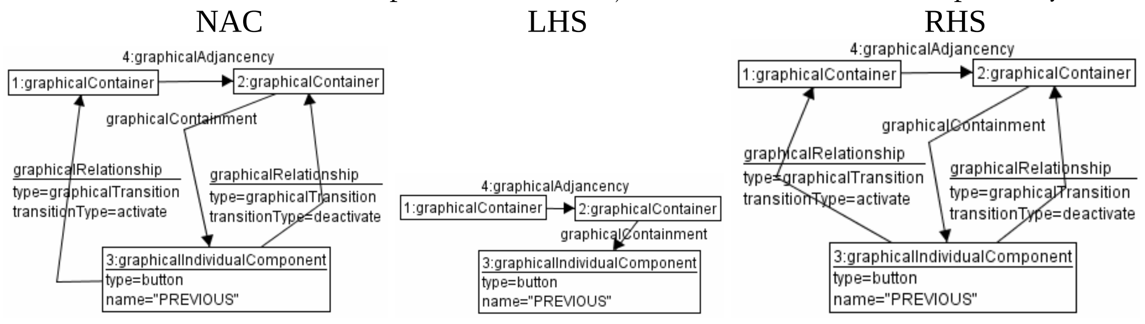


Figure 5-80. Generation of navigation for the “Previous” button

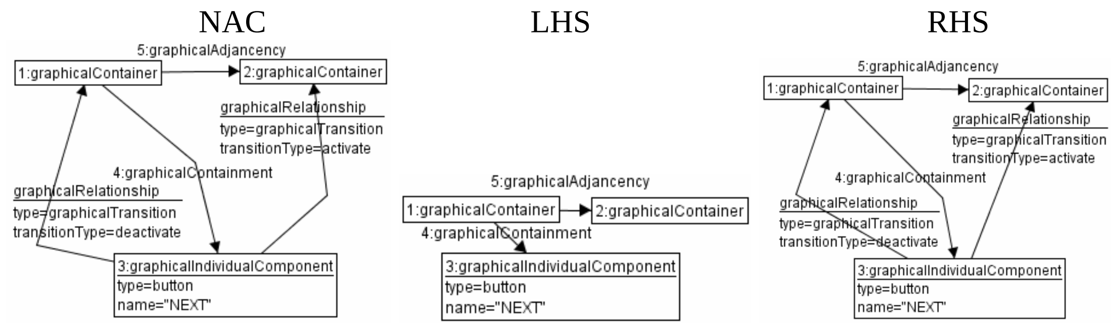


Figure 5-81. Generation of navigation for the “Next” button

Sub-step 3.5: Concrete Dialog Control Definition

The dialog control is ensured by reusing the rules described in Figures 5-31, B-10, B-11 and B-12.

Sub-step 3.6: Derivation of CUI to Domain Relationship

In order to synchronize the GICs with attributes and/or methods of classes from the *Domain Model*, the rules described in Figures 5-32 and 5-33 are reused.

Sub-case 1.2: multimodal CUI presented into separated windows

The present sub-case (T12 in Figure 5-58) contains transformation rules applied on the AUI (separated window case) produced in the previous step, in order to generate the correspondent multimodal CUI.

Sub-step 3.1: reification of AC into CC

The rules used to reify ACs into CCs are described in Figures 5-45 and 5-46. Rule illustrated in Figure 5-45 generates two GC (i.e., the window and the main box of the application) and a VC of type **vocalGroup**. In order to create the group boxes that will embed later the GICs of the UI, rule presented in Figure 5-46 is modified so that the GC of type **box** becomes a **groupBox**, the rest of the rule remaining unchanged.

Sub-step 3.2: Selection of CICs

Based on the multimodal design options defined in Section 4.2.3 we illustrate the rule applied in order to generate the CICs. For each rule we specify in a corresponding table the design options that we take into consideration and their associated values.

For each of the following tasks, **Select pick-up and return information (country, city, day, month, year), Select card type, Select expiration date (month, year)**, we generate a multimodal comboBox widget to which we associate the design options and their CICs concretization described in Table 5-6.

Design option	Value	CIC
Prompting	Multimodal (redundancy)	outputText + vocalMenu
Grouping for input	Multimodal (equivalence)	comboBox + vocalInput
Immediate feedback	Multimodal (redundancy)	comboBox + vocalFeedback
Guidance for input	Iconic (assignment)	imageComponents (keyboard icon +microphone icon)
Guidance for immediate feedback	Iconic (assignment)	imageComponents (speaker icon)

Table 5-6. Design option values for multimodal combobox

The rules that ensure the creation of a **multimodal comboBox** according to the above specified design options are described in the following. In Figure 5-82 the rule generates a GC of type **box** and a VC of type **vocalMenu** that embed, respectively:

- Two GICs: a **comboBox** and an **outputText** (the label associated to the comboBox)
- Two VICs: a **vocalInput** and a **vocalFeedback**.

The guidance for input is ensured by two GICs of type **imageComponent** represented with **keyboard** and **microphone icons**, while the guidance for immediate feedback is associated to the **speaker icon**. Figure 5-83 describes the rule applied in order to create the multimodal items of the comboBox. Thus, for each selection value of an **input facet** of type **select element**, two components will reify it: a GIC of type **item** and a VIC of type **vocalMenuItem**.

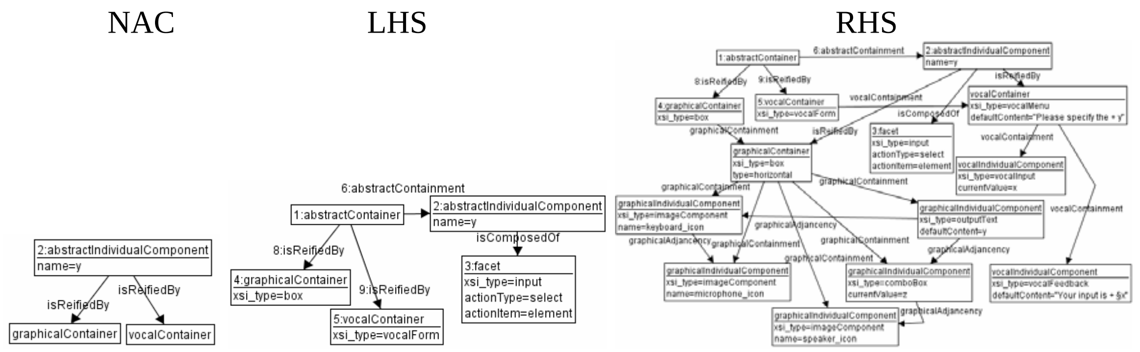


Figure 5-82. Generation of containers that will embed multimodal comboBox items

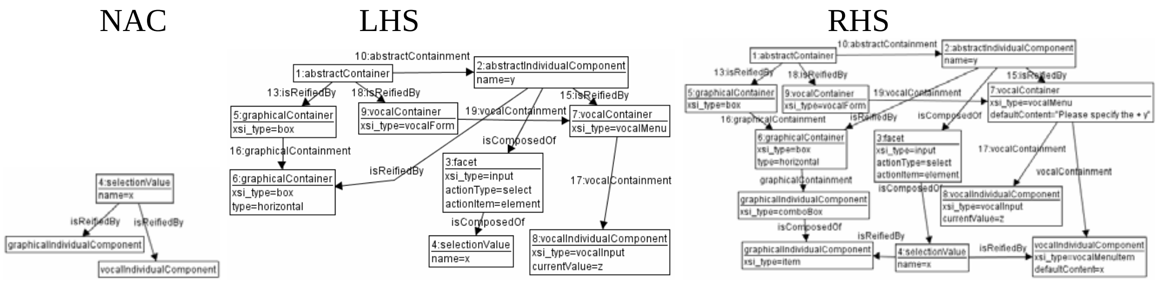


Figure 5-83. Generation of comboBox items and vocalMenuItems for each selection value of a facet of type select

For the **Select car class, Select transmission type, Select insurance type** tasks, we generate multimodal radioButtons to which we associate the design options and their CICs concretization described in Table 5-7.

Design option	Value	CIC
Prompting	Multimodal (redundancy)	outputText + vocalMenu
Grouping for input	Vocal (assignment)	vocalInput
Immediate feedback	Graphical (assignment)	radioButton
Guidance for input	Iconic (assignment)	imageComponents (microphone icon)

Table 5-7. Design option values for multimodal radioButtons

The rules that ensure the creation of multimodal **radioButtons** according to the above specified design options are specified in the following. In Figure 5-84 the rule generates a GC of type **box** and a VC of type **vocalMenu** that embed, repectively a GIC of type **outputText** (the label associated to the group of radioButtons) and a VIC of type **vocalInput**.
The guidance for input is ensured by a GIC of type **imageComponent** represented with a **microphone icon.** Figure 5-85 describes the rule applied in order to create the radioButtons. Thus, for each selection value of an **input facet** of type **select element**, two components will reify it: a GIC of type radioButton and a VIC of type **vocalMenuItem**.

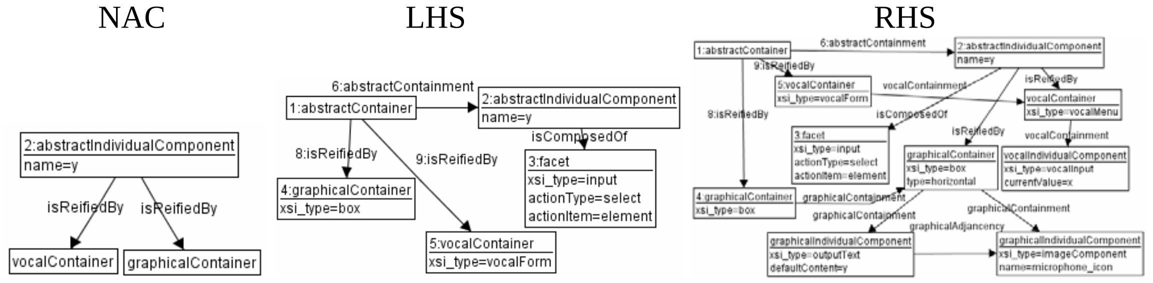


Figure 5-84. Generation of containers that will embed multimodal radioButtons

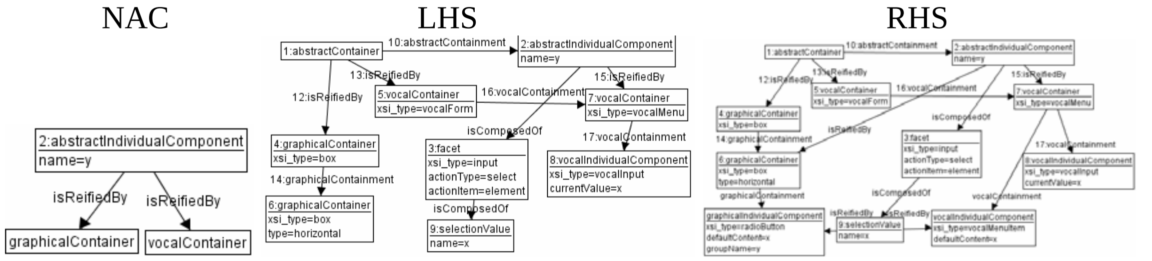


Figure 5-85. Generation of radioButtons and vocalMenuItems for each selection value of a facet of type select

For the **Select optional insurance** task, we generate multimodal checkBoxes to which we associate the design options and their CICs concretization described in Table 5-8.

Design option	Value	CIC
Prompting	Multimodal (redundancy)	outputText + vocalMenu
Grouping for input	Vocal (assignment)	vocalInput
Immediate feedback	Graphical (assignment)	checkBox
Guidance for input	Iconic (assignment)	imageComponents (microphone icon)

Table 5-8. Design option values for multimodal checkBoxes

The rules that ensure the creation of multimodal **checkBoxes** according to the above specified design options are described in the following. In Figure 5-86 the rule generates a GC of type **box** and a VC of type **vocalMenu** that embed, respectively a GIC of type **outputText** (the label associated to the group of checkBoxes) and a VIC of type **vocalInput**. The guidance for input is ensured by a GIC of type **imageComponent** represented with a **microphone icon.** Figure 5-87 describes the rule applied in order to create the checkBoxes. Thus, for each selection value of an **input facet** of type **select element**, two components will reify it: a GIC of type checkBox and a VIC of type **vocalMenuItem**.

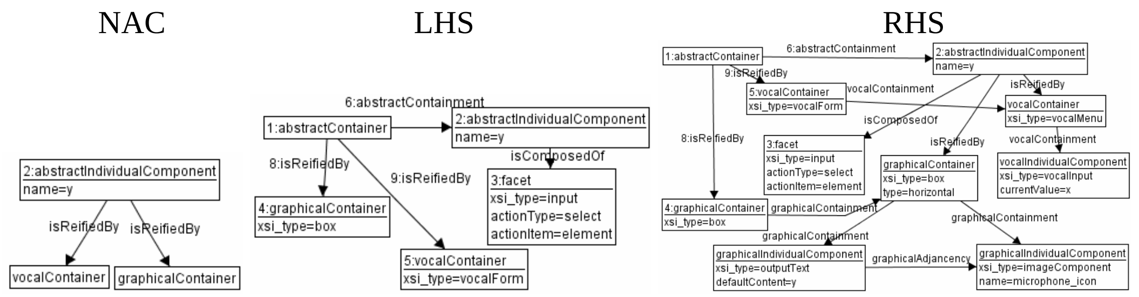


Figure 5-86. Generation of containers that will embed multimodal checkBoxes

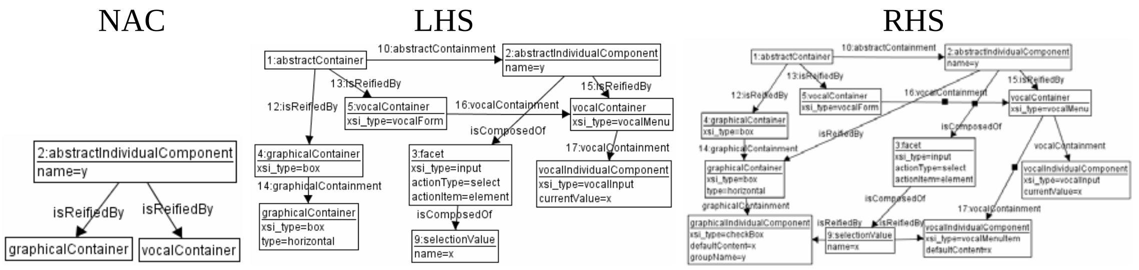


Figure 5-87. Generation of checkBoxes and vocalMenuItems for each selection value of a facet of type select

Table 5-9 presents the design options and their CICs concretization for the **Select car** task for which we generate a listBox widget.

Design option	Value	CIC
Prompting	Graphical (assignment)	outputText
Grouping for input	Graphical (assignment)	listBox
Immediate feedback	Graphical (assignment)	listBox
Guidance for input	Iconic (assignment)	imageComponents (keyboard icon)

Table 5-9. Design option values for listBox widget

In Figure 5-88 the rule generates a GC of type **box** that embed two GICs: a **listBox** and an **outputText** (the label associated to the listBox). The guidance for input is ensured by a GIC of type **imageComponent** represented with a **keyboard icon.** The rule that creates the listBox items is the same as in Figure 5-76.

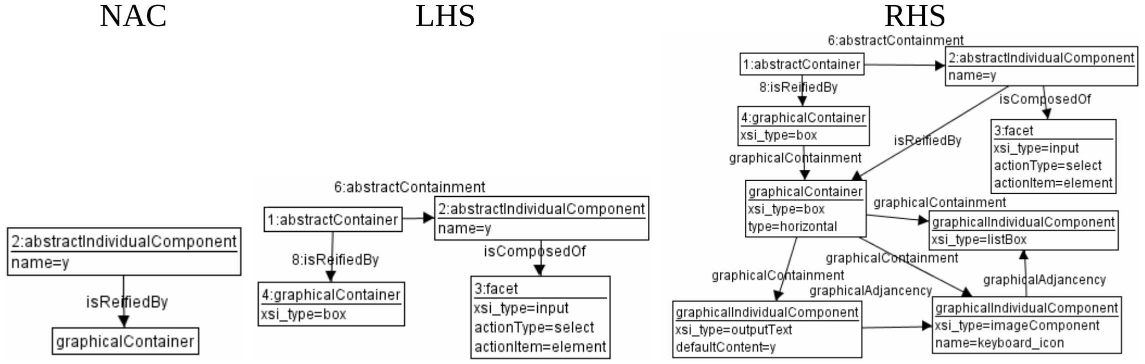


Figure 5-88. Generation of a listBox

Table 5-10 presents the design options and their CICs concretization for the **Input name, Input card number** tasks for which we generate a multimodal **inputText** widget.

Design option	**Value**	**CIC**
Prompting	Multimodal (redundancy)	outputText + vocalPrompt
Grouping for input	Multimodal (equivalence)	inputText + vocalInput
Immediate feedback	Graphical (assignement)	inputText
Guidance for input	Iconic (assignement)	imageComponents (keyboard icon + microphone icon)

Table 5-10. Design option values for multimodal textInput widget

Rule illustrated in Figure 5-89 generates a GC of type **box** that embes two GICs: an **inputText** and an **outputText** (the label associated to the inputText). The rule also generates a VC of type **vocalForm** that embeds two VICs: a **vocalPrompt** and a **vocalInput**. The guidance for input is ensured by two GICs of type **imageComponent** represented with **keyboard and microphone icons.**

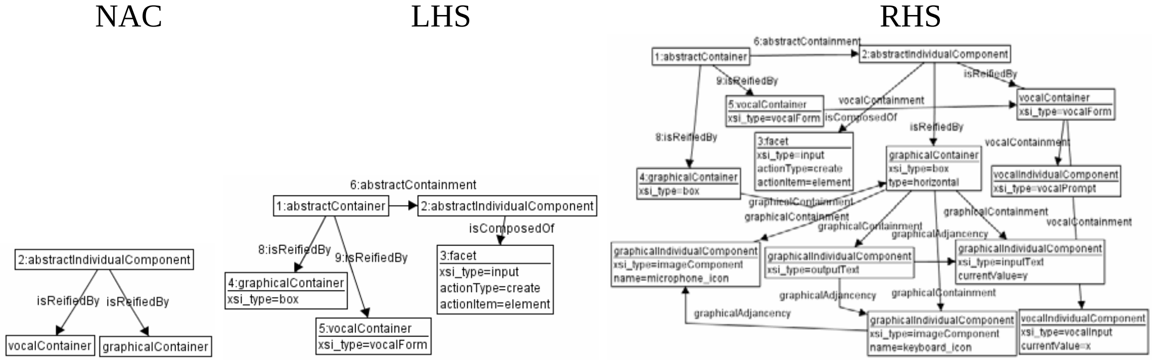


Figure 5-89. Generation of a multimodal inputText

In order to generate the button that allows users to fulfill **Confirm payment** task, we reuse the rule presented in Figure 5-29. For the navigation between the windows we generate the **Next and Previous buttons** by applying the rule from Figure 5-79.

Sub-step 3.3: Synchronization of CICs

This sub-step is applied in order to ensure the synchronization between vocalCIOs and graphicalCIOs generated in the previous sub-step. Thus, between the VIC of type **vocalInput** and the GIC of type **comboBox** generated in Figure 5-82, we generate a synchronization relationship by applying the rule described in Figure 5-90. The synchronization is defined between the currentValue **x** of the VIC and the currentValue **y** of the GIC.

NAC LHS RHS

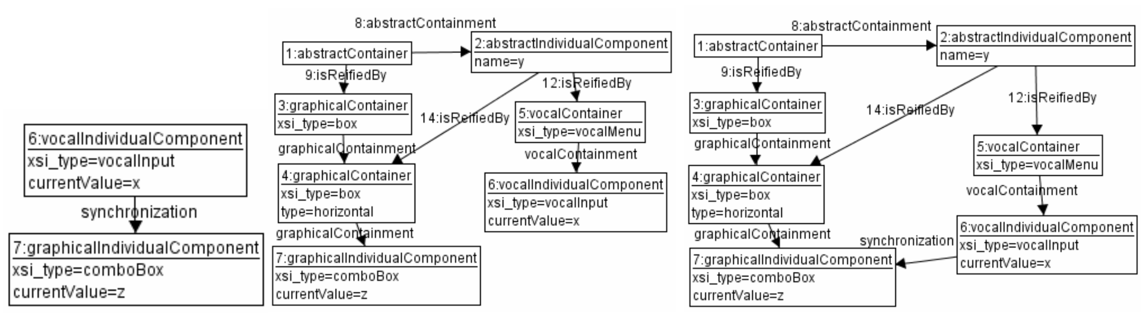


Figure 5-90. Synchronization between a vocalInput and a comboBox

Rule illustrated in Figure 5-91 creates the synchronization relationship between the VIC of type **vocalInput** and GC of type **box** (that embed a group of check boxes) generated in Figure 5-86.

NAC LHS RHS

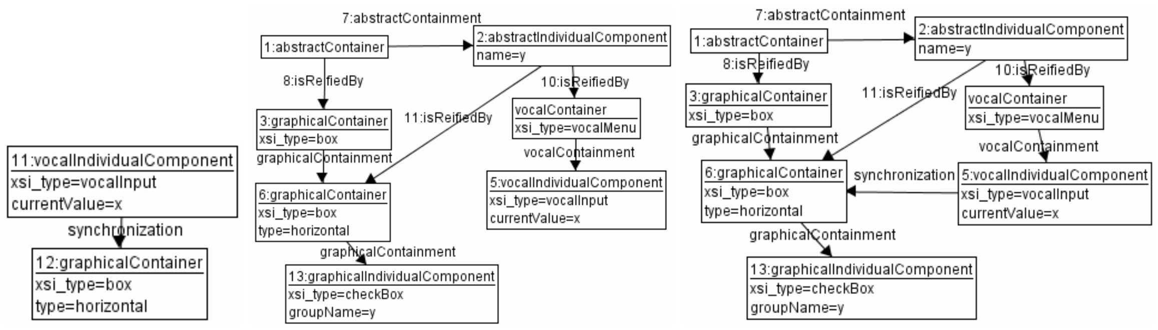


Figure 5-91. Synchronization between a vocalInput and a box that embed a set of check boxes

The synchronization relationship between the VIC of type **vocalInput** and GIC of type **inputText** generated in Figure 5-89 is ensured by applying the rule illustrated in Figure 5-52. For the synchronization relationship between the VIC of type **vocalInput** and GC of type **box** (that embed a group of radio buttons) generated in Figure 5-84 we apply the rule described in Figure 5-53.

Sub-step 3.4: Arrangement of CICs

Rules illustrated in Figures 5-30, B-7, B-8 and B-9 are used to specify the arrangement of graphical CICs. For vocal adjacency relationship, Figures 5-41, B-13, B-14 and B-15 described the rules used to perform the CICs arrangement.

Sub-step 3.5: Navigation definition

The navigation between the windows is specified by the rules illustrated in Figure 5-80 and 81 that endow the **Previous and Next buttons** with **activation/deactivation** power over the adjacent/current GC.

Sub-step 3.6: Concrete Dialog Control Definition

For graphical components the dialog control is ensured by applying the rules described in Figures 5-31, B-10, B-11 and B-12, while Figures 5-42, B-16, B-17 and B-18 are illustrating the rules that define the control of vocal dialog.

Sub-step 3.7: Derivation of CUI to Domain Relationship

In order to synchronize the GICs with attributes and/or methods of classes from the Domain Model, the rules described in Figures 5-32 and 5-33 are reused. For VICs the reused rules are illustrated in Figures 5-43 and 5-44.

Sub-case 2.1: graphical CUI presented into combined group boxes

The present sub-case (T21 in Figure 5-58) contains transformation rules applied on the AUI (combined group boxes) produced in the previous step, in order to generate the correspondent graphical CUI.

Sub-step 3.1: reification of AC into CC

The main window of the UI as well as the GC of type **box** contained in it are generated as a reification of the top most AC (Figure 5-92). Further, each AC contained into the top most AC, is reified into a GC of type **groupBox** (Figure 5-93).

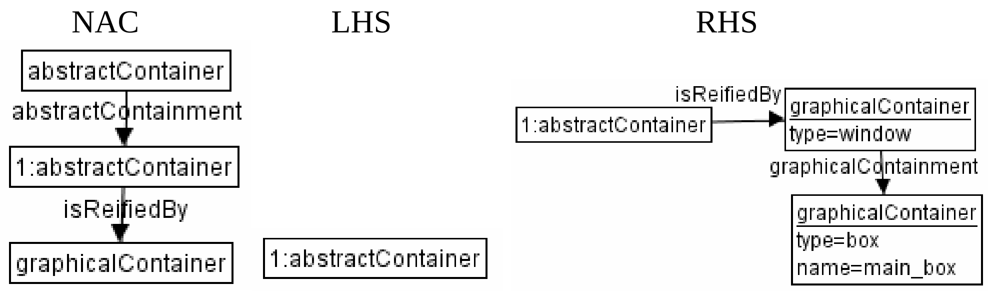


Figure 5-92. Create the main window of the UI

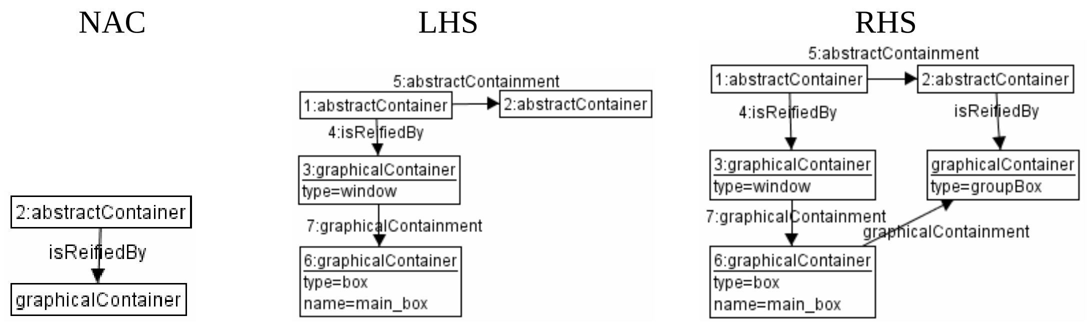


Figure 5-93. Create a group box for each abstract container that belongs to the top-level abstract container

Sub-step 3.2: Selection of CICs

The same set of rules as in the corresponding sub-step of the Sub-case 1.1 will be applied. Nevertheless, as the UI elements are combined in group boxes into one single window, there is no need to generate the navigation buttons. The buttons that allows to confirm/cancel the payment are generated by applying the rule described in Figure 5-29.

Sub-step 3.3: Arrangement of CICs

Rules presented in Figures 5-30, B-7, B-8, B-9 are reused in order to accomplish this sub-step.

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.

Sub-step 3.5: Concrete Dialog Control Definition

The dialog control is ensured by reusing the rules described in Figures 5-31, B-10, B-11 and B-12.

Sub-step 3.6: Derivation of CUI to Domain Relationship

In order to synchronize the GICs with attributes and/or methods of classes from the Domain Model, the rules described in Figures 5-32 and 5-33 are reused.

Sub-case 2.2: multimodal CUI presented into combined group boxes

The present sub-case (T22 in Figure 5-58) contains transformation rules applied on the AUI (combined group boxes) produced in the previous step, in order to generate the correspondent multimodal CUI.

Sub-step 3.1: reification of AC into CC

The rules used to reify ACs into CCs are described in Figures 5-45 and 5-46. By applying these rules we generate the graphical and vocal top most containers that embed a GC of type **box** and a VC of type **vocalForm**, repectively.

Sub-step 3.2: Selection of CICs

In order to generate the CICs for the present sub-case, we reuse the same design options values and consequently the same transformation rules described in the corresponding sub-step of the Sub-case 1.2. However, as the UI elements are combined in grouped boxes into one single window, we do not apply the rule that generates the navigation buttons. The buttons that allows to confirm/cancel the payment are generated by applying the rule described in Figure 5-29.

Sub-step 3.3: Synchronization of CICs

The synchronization between vocalCIOs and graphicalCIOs is ensured by reusing the rules illustrated in Sub-step 3.3 of Sub-case 1.2.

Sub-step 3.4: Arrangement of CICs

Rules illustrated in Figures 5-30, B-7, B-8 and B-9 are used to specify the arrangement of graphical CICs. For vocal adjacency relationship, Figures 5-41, B-13, B-14 and B-15 described the rules used to perform the CICs arrangement.

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.

Sub-step 3.6: Concrete Dialog Control Definition

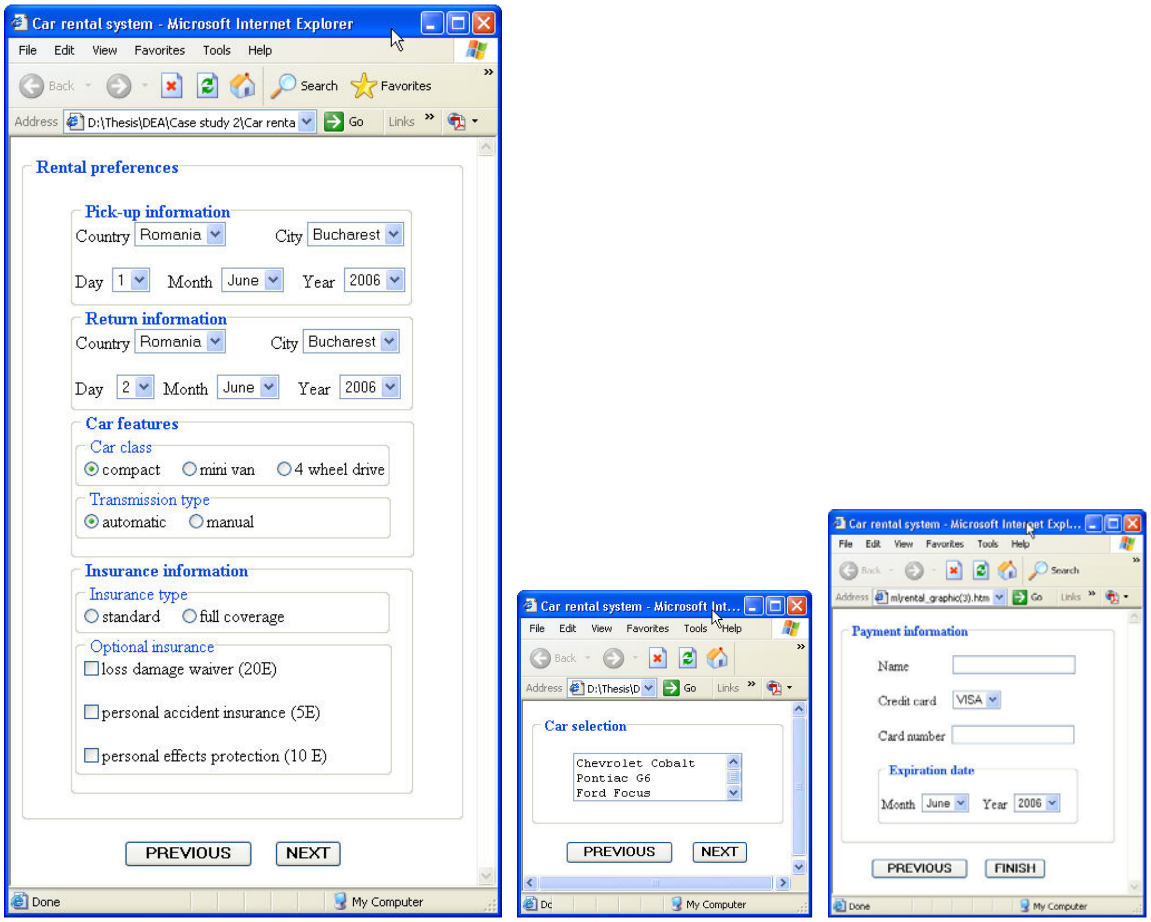
For graphical components the dialog control is ensured by applying the rules described in Figures 5-31, B-10, B-11 and B-12, while Figures 5-42, B-16, B-17 and B-18 are illustrating the rules that define the control of vocal dialog.

Sub-step 3.7: Derivation of CUI to Domain Relationship

In order to synchronize the GICs with attributes and/or methods of classes from the *Domain Model*, the rules described in Figures 5-32 and 5-33 are reused. For VICs the reused rules are illustrated in Figures 5-43 and 5-44.

5.3.3 Step 3: From CUI Model to FUI

This step consists of transforming the CUIs obtained in each of the previous Sub-cases into their corresponding FUI. Thus, once that the specifications for graphical and multimodal the FUIs is produced, we interpret them with a corresponding browser. Figure 5-94 and 5-95 shows the resultant graphical and multimodal FUIs for the Sub-case 1.1 and Sub-case 1.2, respectively, for which the sub-tasks are presented in separated windows. For the Sub-case 2.1 and Sub-case 2.2 that considers both a design option decision for sub-task presentation in combined grouped boxes, we illustrate the resultant graphical and multimodal FUIs in Figures 5-96 and 5-97, respectively.


Car rental system - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address D:\Thesis\DEA\Case study 2\Car renta Go Links
PREVIOUS
NEXT
Done
My Computer
Car rental system - Microsoft Int...
Address D:\Thesis\D
Car rental system - Microsoft Internet Expl...
Address m\rental_graphic(3).htm
FINISH

Rental preferences

Pick-up information
Country: Romania City: Bucharest
Day: 1 Month: June Year: 2006

Return information
Country: Romania City: Bucharest
Day: 2 Month: June Year: 2006

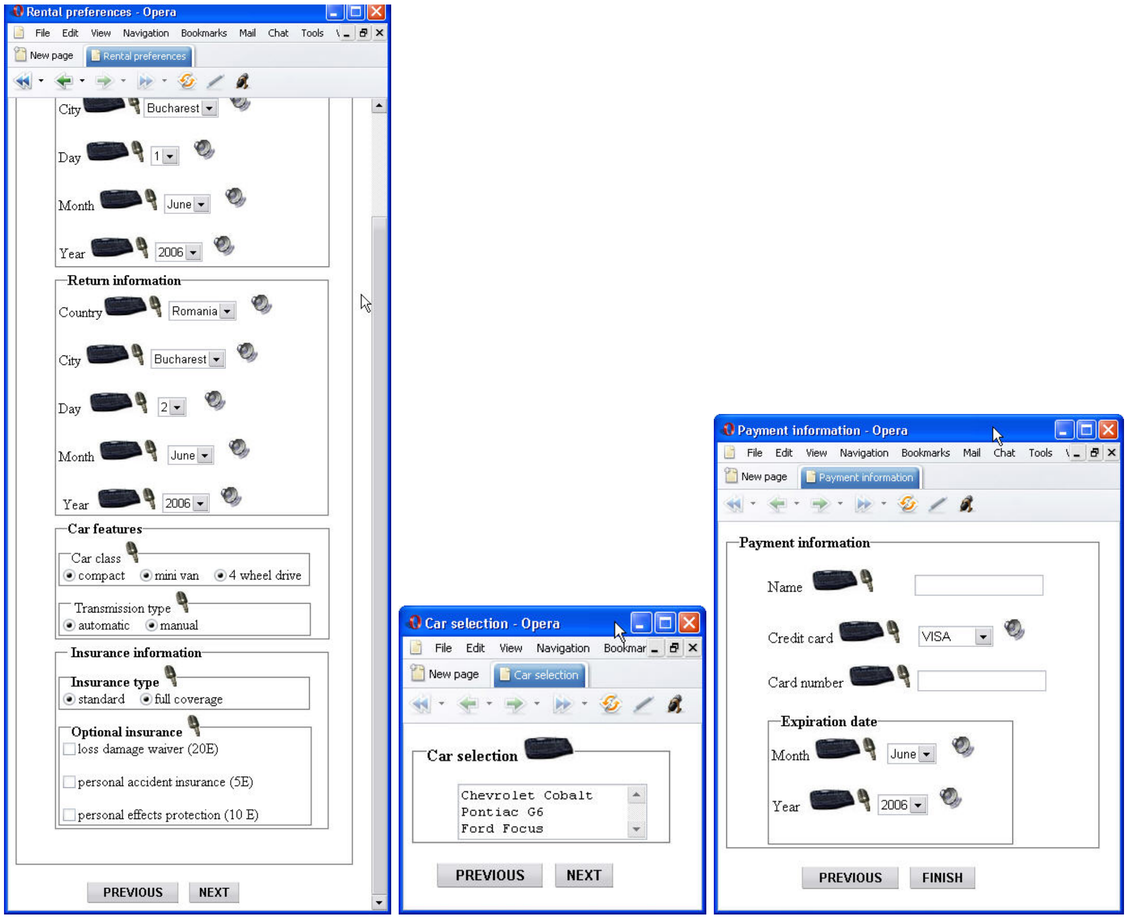
Car features
Car class: [✓ compact] mini van 4 wheel drive
Transmission type: [✓ automatic] manual

Insurance information
Insurance type: standard full coverage
Optional insurance:
- loss damage waiver (20E)
- personal accident insurance (5E)
- personal effects protection (10 E)

Car selection
```
Chevrolet Cobalt
Pontiac G6
Ford Focus
```

Payment information
Name:
Credit card: VISA
Card number:
Expiration date
Month: June Year: 2006

Figure 5-94. Graphical FUI presented into separated windows


Rental preferences - Opera
File Edit View Navigation Bookmarks Mail Chat Tools
New page
Rental preferences
PREVIOUS
NEXT
Car selection - Opera
File Edit View Navigation Bookmar
New page
Car selection
PREVIOUS
NEXT
Payment information - Opera
File Edit View Navigation Bookmarks Mail Chat Tools
New page
Payment information
PREVIOUS
FINISH

City: Bucharest
Day: 1
Month: June
Year: 2006

Return information
Country: Romania
City: Bucharest
Day: 2
Month: June
Year: 2006

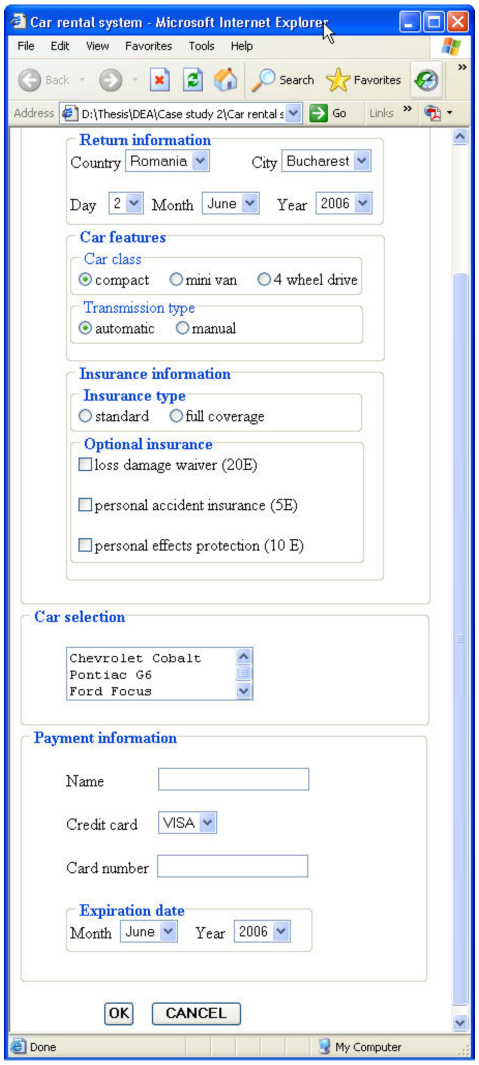
Car features
Car class: [✓ compact] [✓ mini van] [✓ 4 wheel drive]
Transmission type: [✓ automatic] [✓ manual]

Insurance information
Insurance type: [✓ standard] [✓ full coverage]
Optional insurance
☐ loss damage waiver (20E)
☐ personal accident insurance (5E)
☐ personal effects protection (10 E)

Car selection
Chevrolet Cobalt
Pontiac G6
Ford Focus

Payment information
Name:
Credit card: VISA
Card number:
Expiration date
Month: June
Year: 2006

Figure 5-95. Multimodal FUI presented into separated windows


Car rental system - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address D:\Thesis\DEA\Case study 2\Car rental s Go Links
OK CANCEL
Done
My Computer

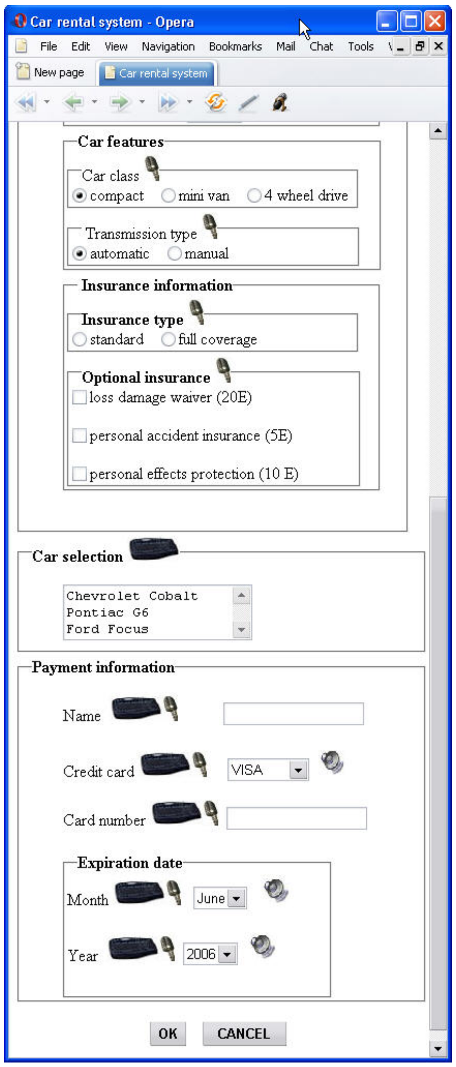
Return information
Country Romania City Bucharest
Day 2 Month June Year 2006

Car features
Car class
[✓ compact] mini van 4 wheel drive
Transmission type
[✓ automatic] manual

Insurance information
Insurance type
standard full coverage
Optional insurance
☐ loss damage waiver (20E)
☐ personal accident insurance (5E)
☐ personal effects protection (10 E)

Car selection
Chevrolet Cobalt
Pontiac G6
Ford Focus

Payment information
Name
Credit card VISA
Card number
Expiration date
Month June Year 2006


Car rental system - Opera
File Edit View Navigation Bookmarks Mail Chat Tools
New page
Car rental system
OK CANCEL

Car features
Car class
[✓ compact] mini van 4 wheel drive
Transmission type
[✓ automatic] manual

Insurance information
Insurance type
standard full coverage
Optional insurance
☐ loss damage waiver (20E)
☐ personal accident insurance (5E)
☐ personal effects protection (10 E)

Car selection
Chevrolet Cobalt
Pontiac G6
Ford Focus

Payment information
Name
Credit card VISA
Card number
Expiration date
Month June
Year 2006

Figure 5-96. Graphical FUI presented into combined group boxes

Figure 5-97. Graphical FUI presented into combined group boxes

5.4 Conclusions

The two case studies presented in this chapter showed the feasibility of our transformational approach for the development of multimodal web user interfaces based on the introduced design options.

CHAPTER 6 CONCLUSION

Chapter 6 concludes this dissertation by differentiating between the stable knowledge (Section 6.1) and the knowledge that was acquired but has to be improved and assessed (Section 6.2). The future work is proposed in Section 6.3.

6.1 Stable knowledge

Multimodal web applications represent a more natural way of communicating with machines. Multimodal UIs capitalizing on the efficiency of visual displays and ease of speech input are overcoming the limitations of current voice browsers and mobile devices. These interactions will thus enable end-users to speak, write, and type, as well as hear and see using a more natural UI than today's single mode browsers.

Multimodal web applications are at the crossroad of two types of applications (Figure 6-1): multimodal applications and web applications. From the set of multimodal web application developed with languages presented in state of the art (see Section 2.2) we have selected X+V as a final target language for our transformational approach. Our choice is motivated by the fact that X+V is an advanced languages for the development of multimodal web user interfaces based on existing, tested standards, which makes it an exceptionally powerful markup language that brings a great deal of versatility to the field of multimodal interface development.

The development of multimodal applications based on X+V language 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 composed of design options that govern design rules encoded as graph grammars which are automated in order to ease the development life cycle of multimodal UIs. Our design options cover only a sub-set of multimodal UIs developed with X+V (see the black circle in Figure 6-1). The rest of multimodal applications represent the difference between the set of UIs developed manually with X+V and the set of UIs obtained automatically by employing our method. The design options cover the properties of three types of web UIs (i.e., graphical, vocal and multimodal). Each design option is described by a corresponding number of values. A set of 19 design options has been identified (Figure 6-2). Each design option has multiple values. The number of possible multimodal UIs that can be generated based on design options are obtained by multiplying the number of values of each design option of the design space. In theory the total number of user interfaces is 388800.

$$\#UI = \prod_{i=1}^{n} D_i \cdot V_{ij} \qquad (\text{j}=1,\ldots,m_i)$$

where,

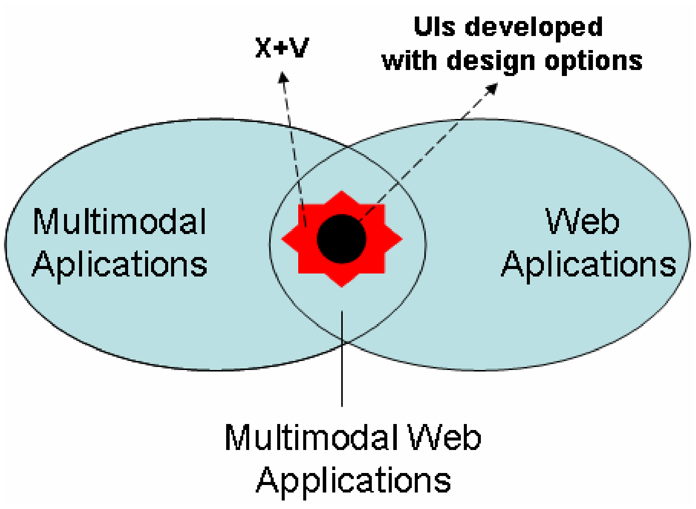
$\#UI$ = number of multimodal UI that can be generated based on design options,

D_i = design option i,

n = 19,

V_{ij} =the value j of the D_i ,

m_i = number of *values* of D_i .


X+V
UIs developed with design options
Multimodal Aplications
Web Aplications
Multimodal Web Applications
Figure 6-1. Positioning of Multimodal Web Applications

The advantage of combining design options into a design space is given by the following three values [Beau00]:

- *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 web UIs and by introspection over the personal knowledge regarding web applications
- *Comparative*: several different designs of multimodal 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, etc.
- *Generative:* the current design space allows discovering potential new design option values on a same axis or introducing new axes associated with yet underexplored design options. This will have a positive impact over the facility of development and quality ergonomics of multimodal applications.

We propose two objectives concerning the design space:

1. **Identification of design values** in order to explore different options for the development of multimodal UIs. Figure 6-2 illustrates the design space considered in the current dissertation. This design space has 19 dimensions where each dimension is denoted by a single design option. By combining the values of different design options we generate a geometrical figure (see the red dashed line that connects different values in Figure 6-2) that corresponds to a possible UI. Here the sub-tasks are presented in separate windows between which we consider a synchronous navigation. The objects that ensure the navigation (multiple cardinality) are separated from the one (single cardinality) charged with the control. Both, the objects that ensure the navigation and the object charged with the control, are placed locally in each window corresponding to each sub-task.

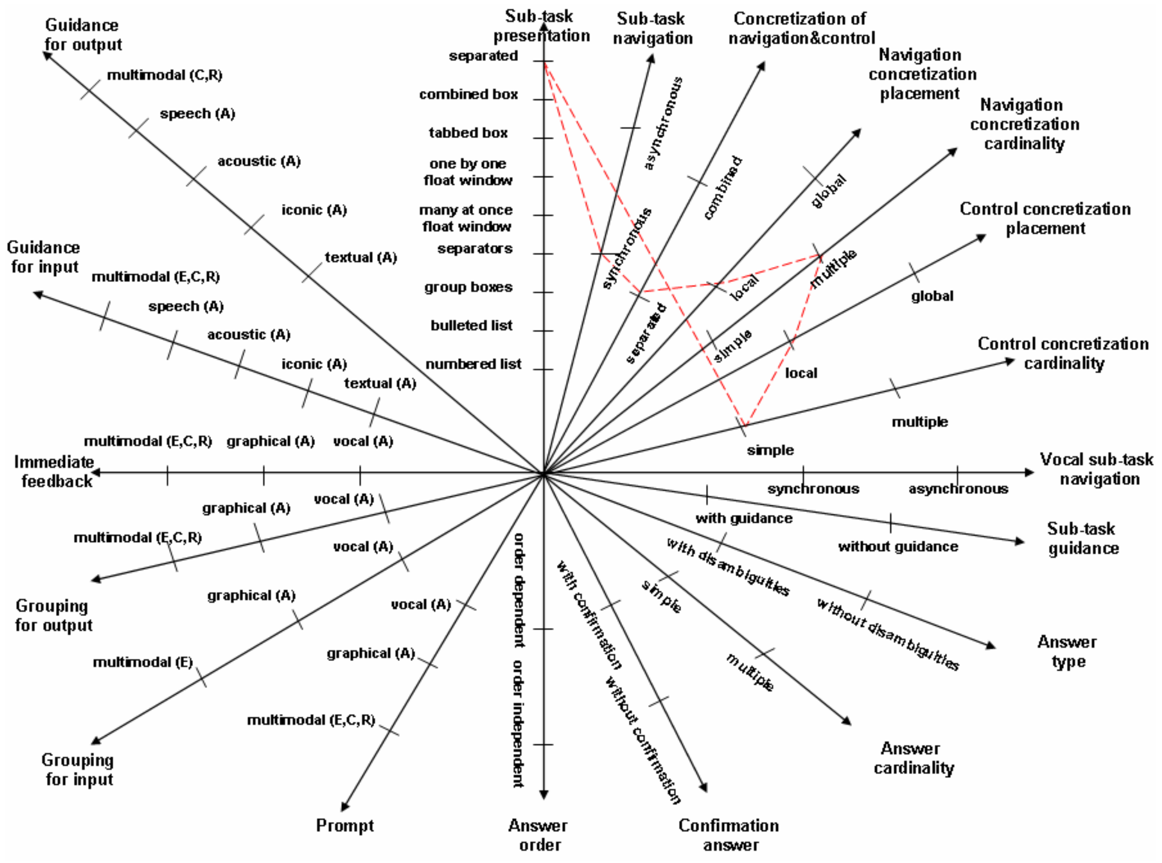


Figure 6-2. Design space for the development of web user interfaces

2. **Identification of dependencies** between the 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 fulfield as dependencies between different design options have been identified. So far we have observed the dependencies illustrated in Table 6-1. The first column specifies the design option values that create the dependencies and the second column the design option values determined by the dependency. Consequently, the total number of user interfaces generated based on our design space (388800) will decrease.

Design option value causing the coupling	Coupled design option value
Sub-task presentation = separated	**Navigation concretization placement** = local
Sub-task presentation = separated	**Control concretization placement** = local
Concretization of navigation & control = combined + **Navigation concretization placement** = local	**Control concretization placement** = local
Concretization of navigation & control = combined	**Navigation concretization placement** = local

+ **Control concretization placement** = local	
Concretization of navigation & control = combined + **Navigation concretization placement** = global	**Control concretization placement** = global
Concretization of navigation & control = combined + **Control concretization placement** = global	**Navigation concretization placement** = global
Concretization of navigation & control = combined + **Navigation concretization cardinality** = multiple	**Control concretization cardinality** = multiple
Concretization of navigation & control = combined + **Control concretization cardinality** = multiple	**Navigation concretization cardinality** = multiple
Grouping for input = graphic (A)	**Immediate feedback** ≠ vocal (A)
Grouping for input = multimodal (E,C,R)	**Immediate feedback** ≠ vocal (A)

Table 6-1. Dependencies between design options

In order to reduce the dependencies between the design options one may consider decomposing the axes that generate the coupling into sub-dimensions corresponding to the coupled values, thus obtaining a design space under the form of a snowflake model. For instance, if we consider the Sub-task presentation as an initial design option (the end-user usualy choses first the type of the presentation for his interface) and we decompose its values with those of the Navigation concretization placement and Control concretization placement, the design space will look like in the Figure 6-3 (for legibility purposes we do not illustrate in the figure the design options for vocal UIs).

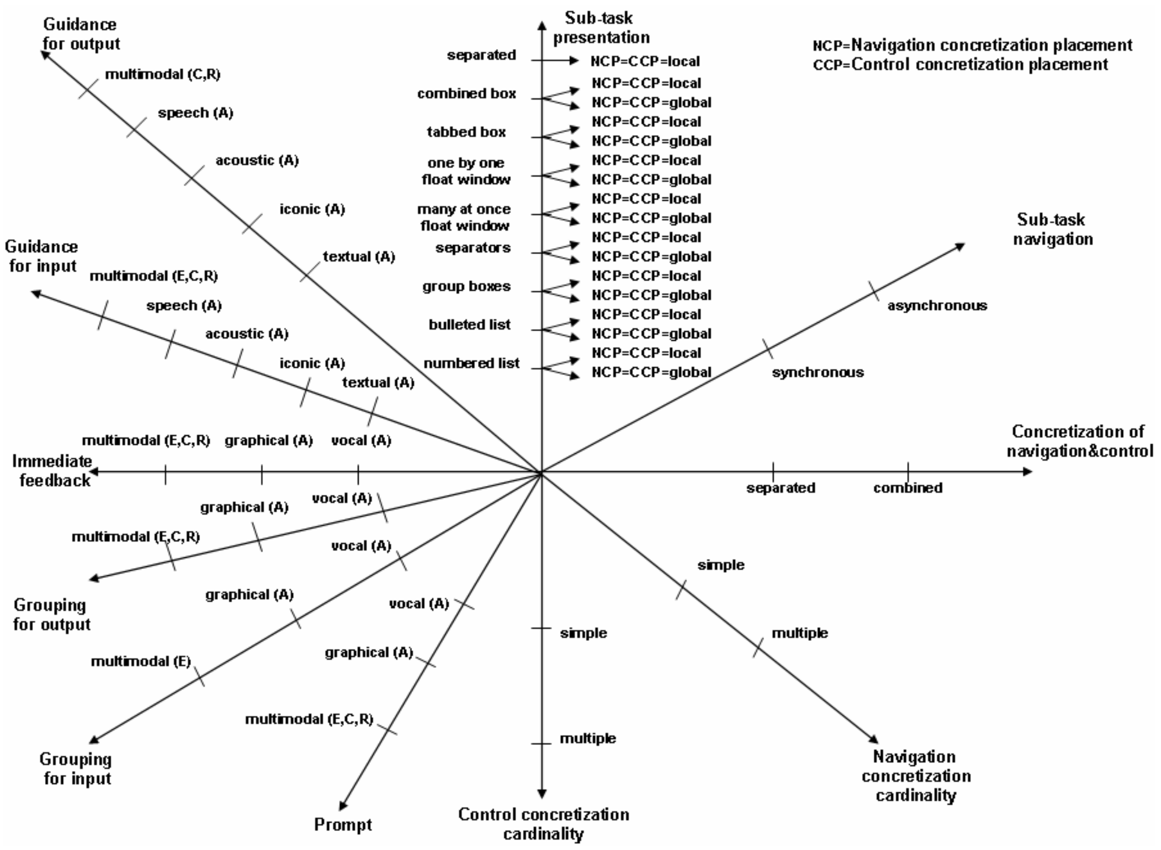


Figure 6-3. Design space under the form of a snowflake model

As it can be observed, the design space is not simplifying, on the contrary it becomes more illisible. Moreover, if we what to reduce the dependencies determined by a group of two design values (see Table 6-1), the representation under the form of a snowflake model will be hard to achieve as the space will become practically imposible to percieve. Due to all these reasons we decided to keep the representation of the design space illustrated in Figure 6-2.

The validation of the current dissertation is achieved by the application of our method on case studies. Their main goal is to show the feasibility of the method i.e., the capability to solve the encountered problems. The two case studies show how design options provide designers with some explicit guidance on what to decide or not for their future user interface, while exploring various design alternatives for the development of graphical, vocal and multimodal UIs. In Section 2.4.2 we identified a set of criterions according to which we analized the limits of the Rush-to-code approach when developing multimoda web user interafaces. Based on the same criterions Table 6-2 compares the result of the application of our method on the two cases studies with the results of the Rush-to-code approach.

Criterion	**Rush-to-code approach**	**Model-based approach**	**Level of impact**
Completeness	No guarantee	High guarantee	End-user
Consistency	No guarentee	High guarantee	End-user
Correctness	No guarantee	High guarantee	End-user
Guidance	No	Yes	End-user
Coding errors	Yes	Low probability	Programmer

Debugging errors	Yes	Low probability	Programmer
Usability errors	Yes	Low probability	End-user
Estimated time for a first development	Novice user (5 days) Expert user (2 days)	Novice user (3 days) Expert user (1/2 day)	Designer
Estimated time for further modifications	Novice user (1/2 day) Expert user (a coulpe of hours)	Novice user (a couple of hours) Expert user (1 hour)	Designer

Table 6-2. Comparison between the Rush-to-code approach and Model-based approach

Our case studies showed the feasibility and the advantages of developing monomodal and multimodal UIs following a model-based approach which relies on explicit transformation catalog at any time. The diversity of the UIs that have been developed in the context of our 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 reuse of the already developed UIs is a consequence of the reuse of transformations that has been illustrated when transformation systems have been straightforwardly reused from one case study to another or from one sub-case to another. Thus, we avoid the development of transformation catalogs that are applied only for a particular case study and consequently we try to prevent the proliferation of UIs that are close to each other.

In order to support the model-driven approach proposed in the current work for the development of multimodal web user interfaces we have considered UsiXML, a user interface descriprtion language. The advantage of our approach consists in the fact that the conceptual part can be incorporated in other languages like XIML, UIML, etc. However, we do not have the guarantee that the Consortium which is in charge with the considered language will take into account our proposition.

The assessment of the characteristics of our methodology is based on a set of selected criteria, called requirements. These requirements were presented in Section 2.4.2 based on the state of the art of current monomodal/multimodal language. A part of the requirements were already fulfilled by the UsiXML language [USIX05] which created the motivation for choosing it as a basis for the extension to multimodal UIs development. The rest of the requirements are discussed in the following:

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

Discussion: The current version of UsiXML [USIX06] supports the development of UIs which allows multimodal input interaction by synchronizing graphical CIOs (see Section 3.2.1) and vocalCIOs (see Section 3.2.2).

Requirement 2. Support for multimodal output: states that our ontology should allow multiple (i.e., at least two different) output interaction modalities. The requirement is motivated by the definition of the multimodal systems (see Section 1.3.4).

Discussion: The current version of UsiXML [USIX06] supports the development of UIs which allows multimodal output interaction by synchronizing graphicalCIOs (see Section 3.2.1) and vocalCIOs (see Section 3.2.2).

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

Discussion: due to the fact that our target is X+V language (see Section 4.3), Complemenarity property is not supported neither in input nor in output as it requires data fussion and data fission, repectively. Fusion and fission of data are not supported by the X+V multimodal browsers. Thus, our ontology ensures the support for the following properties:

- *Assignment*: only the involved type of CIO (i.e., the graphicalCIO or the vocalCIO) will be generated in order to allow the input from the user
- *Equivalence*: by the introduction of the *synchronization* relationship (see Section 3.2.4) which allows synchronizing vocal input with the associated graphical component.
- *Redundancy*: both, the graphicalCIOs and vocalCIOs will be generated in order to allow the redundant input of the user

Requirement 4. CARE properties support for output modalities: states that our ontology should ensure the support of the CARE properties for output modalities. The current requirement is motivated by the design facilities offered by the CARE properties when defining the relationships that can occur between output modalities.

Discussion: due to the fact that our target is X+V language (see Section 4.3), Complemenarity property is not supported neither in input nor in output as it requires data fussion and data fission, repectively. Fusion and fission of data are not supported by the X+V multimodal browsers. Thus, our ontology ensures the support for the following properties:

- *Assignment*: only the involved type of CIO (i.e., the graphicalCIO or the vocalCIO) will be generated in order to ensure the output of the system
- *Redundancy*: both types of CIOs (i.e., the graphicalCIO or the vocalCIO) will be generated in order to ensure the output of the system.
- *Redundancy*: both, the graphicalCIOs and vocalCIOs will be generated in order to allow the redundant output of the system

Requirement 5. Approach based on design space: states that our development life cycle towards a final multimodal web user interface should be supported by a set of design features. This requirement is motivated by the need to clarify the development process in a structured way in terms of options, thus requiring less design effort.

Discussion: This requirement is completely fulfilled by the introduction in the development life cycle of a set of design options for graphical and multimodal UIs (see Section 4.2).

Requirement 7. Separation of modalities: states that the concepts and the specifications corresponding to each modality should be syntactically separated one from each other. The current requirement is motivated by two aspects: (1) the flexibility in developing applications due to the fact that the specifications for each modality can be developed separately from the other modalities specifications and combined them altogether later, (2) reusability of the specification or part of a specification of a modality in other applications that involve the use of the same modality.

Discussion: the separation of the concepts and specification corresponding to each modality is fully achieved and illustrated in Sections 3.2 and 3.3.2, respectively. The *synchronization* relationship is employed in order to ensure the synchronization between the involved modalities when needed.

Requirement 15. Methodological 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 hinder designers to add, delete, modify and reuse these steps.

Discussion: [Limb04] identifies the development sub-steps for graphical and vocal UIs. We reuse these sub-steps for multimodal UIs but we also add a new one in order to logically support the development process of multimodal UIs i.e., the *synchronization between CICs*. This sub-step is ensuring the coordination of vocalCIOs and the graphicalCIOs by generating a synchronization relationship between them.

6.2 Knowledge acquired to be improved and assessed

The current dissertation is located in the discipline of Engineering for Human-Computer Interaction (EHCI). This discipline founds itself at the intersection of two other disciplines (Figure 6-4): Software engineering (SE) and Human-Computer Interaction (HCI). SE can be defined as the application of a systematic, disciplined, quantifiable approach to development, operation, and maintenance of software, more exactly, the application of engineering to software [IEEE90]. HCI can be defined as a discipline concerned with design, evaluation and implementation of interactive computing system for human use and with the study of major phenomena surrounding them [Hewe96].

Cognitive psychology is the psychological science that studies knowledge and the mental processes that underlie behavior, including decision making, thinking and reasoning. Cognitive psychology covers a broad range of research domains between which the discipline of Engineering for Human-Computer Interaction (EHCI). Decision making, as a feature of the cognitive psychology, plays an important role in the area of EHCI 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 in order to offer detailed guidance for designers to use during the development process. While the cognitive psychology offers a support for the usability of the decision making, SE and HCI published studies for multimodal UI design, a subset of the UIs considered by each of the two disciplines, are surprisingly rare. Furthermore, there are few ongoing works on usability of multimodal user interfaces mainly because there are not so many multimodal applications. There have been a number of studies (e.g., [Lars03a]) of the way designers should conceptualize their multimodal 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 multimodal applications still remains an uncovered research area as there are no usability multimodal applications experiments for this.

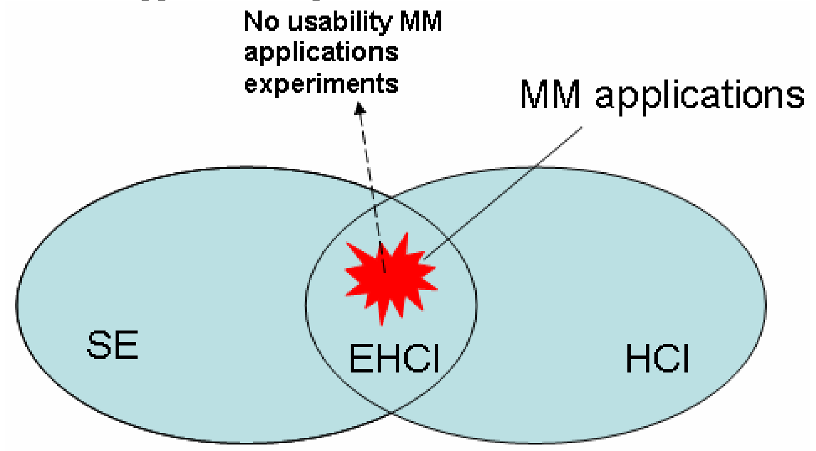


Figure 6-4. Positioning MM applications at the crossroad of SE and HCI

The present dissertation is focused on the feasibility of code generation, while the usability of the UIs is let to be done as an internal validation of the Ph.D. thesis associated to the present dissertation. It will consist in a study made over the qualitative principles derived from cognitive psychology and their applicability to the design options for multimodal web applications presented in the current work.

6.3 Remaining knowledge

There is a lot of work to do around the framework presented in the current dissertation. In the following we emphasize which are the most important issues for us to solve in the future work:

- **Investigate the implementation of a new transformation engine.** The current work proposes a model-driven approach for developing multimodal web UIs based on a framework that structures the development life cycle into four levels of abstractions. Each level relies on corresponding UsiXML meta-models defined in terms of UML class diagrams (see Section 3.1). Instances of different meta-models ensure different view points of UIs (e.g., *Abstract Models* are modality-independent, *Concrete Models* are toolkit-independent). In order to pass from one view point to another (e.g., from *Abstract Model* considered *Source Model* to *Concrete Model* considered *Target Model*), UsiXML specifies a *Transformation Model* (Figure 6-5) which is an instance of the *UsiXML Transformation Meta-Model* (see Section 3.1.8). The *Transformation Model* specifies a set of transformation rules taken from a transformation catalog (see Section 4.1). The *Source Model* (e.g., *Abstract Model*) and the *Transformation Model* are transformed into graph structures. Further, the *Source Graph* is submitted to a set of *Graph Transformations* in order to generate the *Target Graph* which is transformed into the *Target Model* (e.g., *Concrete Model*). Finally, the *Target Model* is checked upon its meta-model.

  Model transformation is a relatively unexplored research area. The use of graphical modeling language and the application of object-oriented metamodeling to language definition is setting a new context for exploring model transformations. Many new approaches to model-to-model transformation have been proposed, but little experience is available to asses their effectiveness in practical applications. The available modeling tools are just starting to offer some model-to-model transformation capabilities, but these are still very limited and without proper theoretical foundation. An initiative in the area of model-to-model transformation is the Query/Views/Transformations (QVT) [QVT05] provided by Object Management Group (OMG) which aims at creating a standard approach for model transformations in the context of Model Driven Architecture (MDA) framework. MDA [Mill03] is an OMG standard that tries to automate the generation of platform-specific models from platform-independent models. QVT is a language that allows defining transformations between models which are instances of meta-models described according to Meta-Object Facility (MOF) specification. MOF specification is an OMG standard that defines an abstract language and a framework for specifying, constructing, and managing technology neutral metamodels, such as UML technology. The transformations are also defined according to a *Transformation Model* (Figure 6-5) which is an instance of a *Transformation Meta-Meta-Model* defined according to MOF specification. Due to the fact that UsiXML Meta-Models are defined in terms of UML class diagrams and that the framework presented in the current dissertation has close similarities with the MDA approach, we propose for the future work to asses the applicability of QVT for UsiXML mode-to-model transformations. This will suppose that an instance of a *Source Meta-Model* (e.g., *Abstract Model*) is submitted to a set of transformation defined according to the QVT language in order to generate an instance of the *Target Meta-Model* (e.g., *Concrete Model*. The advantage of this approach is given by the fact that it employs the same development steps and sub-steps illustrated in the current work, which makes our methodology independent of the transformation engine.

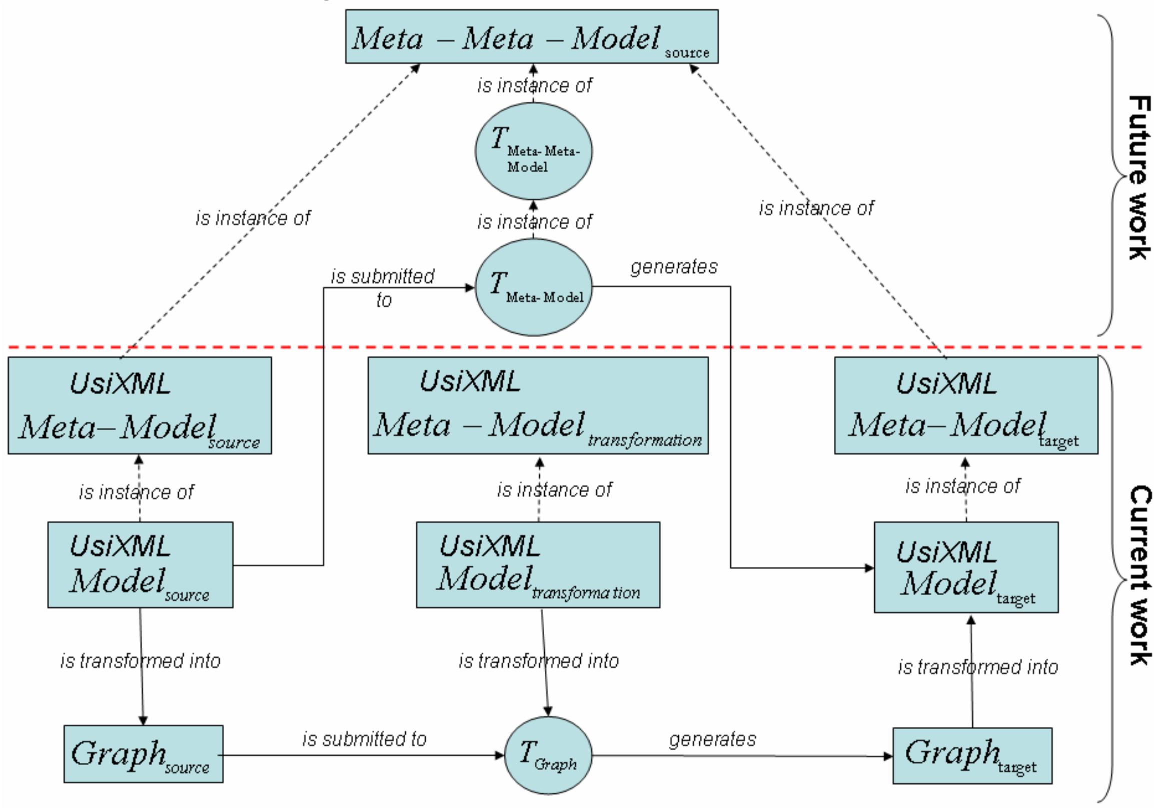


Figure 6-5. Mode-to-model transformations

- **Reinforce existing vocal components.** The current ontology proposed a basic set of vocal components in order to prove the feasibility of our approach. For future work we propose to add new features to the already existing vocal components which will assure more complete interaction between the end-user and the system.
- **Extend the ontology with new vocal components.** The current work is based on a set of vocalCIOs that ensure the basic functionality for vocal and multimodal applications. In order to ensure the development of more complex types of UIs, new vocal components have to be considered.
- **Design space improvement.** We may want to perform the following activities over the design space introduced in the current work: (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 transformation catalogs.** The transformational approach presented in the context of this dissertation suffers from an intrinsic incompleteness. We have applied a series of transformations systems in order to generate different types of UIs (see Section 5), but the same set of rules would not prove to be suitable if the case studies are modified. For future work we propose to enrich the transformation catalogs with new transformation system in order to cover more types of UIs.
- **Adaptation of multimodal web user interfaces to the context of use.** In order to respond to the modifications of the three parameters considered by the context of use (i.e., user, platform, environment), we consider for future work the translation from a source multimodal UI to a target multimodal UI. Thus we will generate adaptable/adaptive multimodal UIs. This translation may imply changes in the set of the supported modalities and/or changes in the way the CARE properties are supported.
- **Enlargement of multimodal user interface application area.** So far we took into consideration user interafaces where the end-used had to deal with input and output tasks in form type applications. Other types of applications will be considered in the future work, such as applicatios that imply multimodal commands for the navigation over 2D satellite images.

REFERENCES

A

[Abra04]

Abrams, M., Helms, J., *User Interface Markup Language (UIML) Specification Working Draft 3.1*, 11 March 2004. Available at: 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 OOPSLA'03, October 26 - 30, 2003, Anaheim, California, USA.

[Aneg04]

Anegg, H., Niklfeld, G., et al., *Multimodal Interfaces in Mobile Devices – The MONA Project*, Workshop paper for the MobEA II - Emerging Applications for Wireless and Mobile Access Workshop, New York City, USA, May 18th 2004.

[Anno01]

Anoop K. Sinha, Scott R. Klemmer, Jack Chen, James A. Landay, Cindy Chen, *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.

B

[Bast93]

Bastien, J. M. C., Scapin, D.L., *Ergonomic Criteria for the Evaluation of Human-Computer Interfaces*, Institut National de Recherché en Informatique et en Automatique, France, 1993.

[Beau00]

Beaudouin-Lafon, M., *Instrumental Interaction: An Interaction Model for Designing Post-WIMP User Interfaces*, Proc. ACM Human Factors in Computing Systems CHI'2000 (The Hague, 1-6 April 2000), ACM Press, New York, 2000, pp. 446-453.

[Bleu04]

Bellik, Y., Teil, D., *Les types de multimodalities*, 4ème journées sur l'inginierie des Interface Homme-Machine, Acte IHM, 1992.

[Bleu04]

Bleul, S., Mueller, W., Schaefer, R., *Multimodal Dialog Description for Mobile Devices*, Proceedings of the "Developing User Interfaces with XML: Advances on User Interface Description Languages" Conference, Workshop of the AVI'04, Gallipoli, Italy, 2004, pp.119-126.

[Bolt80]

Bolt, R.A., *Put-that-there: Voice and gesture at the graphics interface,* International Conference on Computer Graphics and Interactive Techniques, Proceedings of the 7th annual conference on Computer graphics and interactive techniques, Seattle, Washington, United States, pp. 262 – 270, 1980.

[Bouc04a]

Bouchet, J., Nigay, L., Ganille, T., *ICARE Software Components for Rapidly Developing Multimodal Interfaces,* Proceedings of the 6th international conference on Multimodal interfaces, State College, PA, USA , 2004, pp. 251 - 258

[Bouc04b]

Bouchet, J., Nigay, *ICARE: A Component-Based Approach for the Design and Development of Multimodal Interfaces*, Proceedings of ACM-CHI'04, Extended Abstracts, Vienna, Austria, April 2004, ACM Press, pp. 1325-1328.

C

[Calv03]

Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., and Vanderdonckt, J., A Unifying Reference Framework for Multi-Target User Interfaces. Interacting with Computers 15(3), June 2003, pp. 289–308.

[Cohe98]

Cohen, P. R., Johnston, M. et al., *The Efficiency of Multimodal Interaction: A Case Study*, International Conference on Spoken Language Processing, ICSLP'98, Australia, 1998, pp. 249-252.

[Cole85]

Cole, I., Lansdale, M., Christie, B., Dialogue design guidelines, In B. Christies (Ed.), Human Factors Of Information Technology In The Office, 1985, Chichester, UK: Wiley, In Lansdale, M.W., & Ormerod, T.C. (1994) Understanding interfaces: a handbook of human-computer dialogue, Academic Press, UK.

[Cout92]

Coutaz, J., "Multimedia and Multimodal User Interfaces: A Software Engineering Perspective", International Workshop on Human Computer Interaction, 1992, St Petesburg.

[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 the INTERACT'95 conference, S. A. Arnesen & D. Gilmore Eds., Chapman&Hall Publ., Lillehammer, Norway, June 1995, pp. 115-120

[Czar03]

Czarnecki K., Helsen, S., Clasification of Model Transformation Approaches, in proceedings of the Workshop on Generative Techniques in the Context of Model-Driven Architecture, OOPSLA’03.

D

[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. (eds.), *Handbook of Graph Grammars and Computing by Graph Transformation, Application, Languages and Tools*, Vol. 2, World Scientific, Singapore, 1999.

H

[Hewe96]

Hewett T. T., Baecker R., Carey T., Gasen J., Mantei M., Perlman G., Strong G., and 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.

I

[IBM05]

IBM*, WebSphere Voice Toolkit Getting Started Version 6.0., Second Edition, November, 2005.* Available at: 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.

K

[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, British Columbia, Canada, 2003, pp.281-284.

[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, pages 187-212, November 2003.

L

[Lars03a]

Larson, J., A., *Commonsense Guidelines for Developing Multimodal User Interfaces*, Larson Technical Services, 3 April, 2003. Available at: 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 at: http://www.w3.org/TR/mmi-framework/

[Limb00]

Limbourg, Q., Vanderdonckt, J., and Souchon, N., *The Task-Dialog and Task-Presentation Mapping Problem: Some Preliminary Results*, in F. Paternò, Ph. Palanque (eds.), Proc. of 7th Int. Workshop on Design, Specification, Verification of Interactive Systems DSV-IS'2000 (Limerick, 5-6 June 2000), 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*, in Proceedings of the 5th International Conference on Computer-Aided Design of User Interfaces CADUI'2004, Madeira, January, 14-16, 2004), Kluwer Academics Publishers, Dordrecht, 2004.

[Limb04b]
Limbourg, Q., *Multi-Path Development of User Interfaces*, PhD thesis, University of Louvain, November, 2004.

M

[Macl89]
MacLean, A., Young, R., and Moran, T., *Design rationale: the argument behind the artifact*, Proc. of the ACM Conf. on Human Aspects in Computing Systems CHI'89 (Austin, 30 April-4 May 1989), ACM Press, New York, 1989, pp. 247-252.

[Maes03]
Maes, S., H., Saraswat, V., *Multimodal Interaction Requirements*, W3C Note 8 January 2003. Available at: http://www.w3.org/TR/mmi-reqs/#Inputmodalityrequirements

[Mars87]
Marshall C., Nelson C., Gardiner M.M., *Design guidelines. In Applying Cognitive Psychology to User- Interface Design*, eds. M. M. Gardiner and B. Christie, Chichester:Wiley & Sons Ltd, 1987.

[Mart02]
Martin, J.-C., Kipp, M., *Annotating and Measuring Multimodal Behaviour - Tycoon Metrics in the Anvil Tool*, *Proceedings of the 3rd International Conference on Language Resources and Evaluation (LREC'2002)*, Las Palmas, Canary Islands, Spain, 29-31 may 2002 (Martin_LREC'02.pdf)

[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.

[Mart97]
Martin, J. C., *TYCOON: Theoretical Framework and Software Tools for Multimodal Interfaces*, Intelligence and Multimodality in Multimedia Interfaces, AAAI Press, 1997.

[Mell03]
Mellor S. J., and Clark A. J. (Eds.), *Introduction to Model Driven-Development*, in IEEE Software 20(5), 2003, pp. 14-18.

[Meri06]
Merisol, M., Badia, F., *Evaluation de la maquette d'un service multimodal de recherché d'itinéraire dans un reseau de bus*, Proceedings of the 18th international conference on Association Francophone d'Interaction Homme-Machine,Montreal, Canada, pp. 241– 244, 2006.

[Mill03]
Miller, J., Mukerji, J., *MDA Guide Version 1.0.1,* Available at: 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

N

[Niel88]
Nielsen., J., Coordinating user interfaces for consistency, Workshoop held during the 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, numéro spécial Multimédia et Collecticiel,Volume 15, N° 9, 1996, AFCET&Hermes Publ, 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 factors in computing systems, Denver, Colorado, United States, Pages: 98 – 105, 1995, ISBN:0-201-84705-1.

[Niga97b]
Nigay, L., Coutaz, J., *Multifeature Systems: The CARE Properties and Their Impact on Software Desig*n, 1997, Intelligence and Multimodality in Multimedia Interfaces: Research and Applications, AAAI Press Publ. CD-ROM et Web. J. Lee Ed, 1997.

[Niga97c]
Nigay, L., Coutaz, J., A design space for multimodal systems: Concurrent processing and Data fusion, in proceedings of the 12 th BCS conference on Human Computer Interaction, HCI'97, Springer Verlag.

O

[Ovia99]
Oviatt, S., *Ten myths of multimodal interaction*, Communications of the ACM, Volume 42, Issue 11, November 1999, pp.: 74 – 81, ISSN:0001-0782, ACM Press, New York, USA.

P

[Pala03]
Palanque, Ph. and Schyn, A., *A Model-Based Approach for Engineering Multimodal Interactive*, Proc. of 9th IFIP TC13 Int. Conf. on Human-Computer Interaction Interact'2003 (Zurich, 1-5 September 2003), IOS Press, Amsterdam, 2003, pp. 543-550.

[Pate97]
Paternò F. , Mancini C. , and Meniconi S., ConcurTaskTree: A diagrammatic notation for specifying task models, in Howard S. , Hammond J. , and Lindgaard G. (Eds.), Proceedings of IFIP TC 13 International Conference on Human-Computer Interaction Interact'97 (Sydney, July 14-18, 1997), Kluwer Academic Publishers, Boston, 1997, pp. 362-369.

[Puer02a]
Puerta, A., Eisenstein, J., XIML: A Common Representation for Interaction Data, Sixth International Conference on Intelligent User Interfaces IUI2002, January 13-16, 2002, San Francisco, USA, ACM Press, pp. 214-215.

[Puer02b]
Puerta, A., Eisenstein, J., *XIML: A Universal Language fou 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.

R

[Rous05]
Rousseau, C., Bellik, Y., and Vernier, F., *Multimodal output specification/simulation platform*, Proc. of 7th Int. Conf. on Multimodal Interfaces ICMI'2005, Trento, 4-6 October 2005, ACM Press, New York, 2005, pp. 84-91.

S

[Scot00]
Scott R. Klemmer, Anoop K. Sinha, Jack Chen, James A. Landay, Nadeem Aboobaker, and Annie Wang, *"SUEDE: A Wizard of Oz Prototyping Tool for Speech User Interfaces."* in *CHI* Letters, The 13th Annual ACM Symposium on User Interface Software and Technology: UIST 2000. 2(2): p. 1-10.

[Schy05]
Schyn, A, Une approche fondée sur les modèles pour l'inginérie des systèmes interactif multimodaux, Thèse de doctorat, Université Toulouse III, 2005.

[Stan05]
Stanciulescu, A., Limbourg, Q., Vanderdonckt, J., Michotte, B., Montero, F., *A Transformational Approach for Multimodal Web User Interfaces based on UsiXML*, Proc. of 7th Int. Conf. on Multimodal Interfaces ICMI'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 Applications,* Proc. of 6th International Conference on Computer-Aided

Design of User Interfaces CADUI´2006, Bucharest, Romania, 6-8 June, 2006, Chapter 4, Springer-Verlag, Berlin, 2006, pp. 41-56.

[Svgo99]
Scalable vector graphics. Overview. Available at: http://www.adobe.com/svg/overview.html

U

[USIX06]
UsiXML Consortium, *UsiXML, a General Purpose XML Compliant User Interface Description Language*, UsiXML V1.6.4, 1 March 2006. Available at http://www.usixml.org/index.php?view=page&idpage=6

[USIX05]
UsiXML Consortium, *UsiXML, a General Purpose XML Compliant User Interface Description Language*, UsiXML V1.6.3, 16 June 2005. Available at http://www.usixml.org/index.php?view=page&idpage=6

V

[Vand01]
Vanderdonckt, J., Bouillon, L., and Souchon, N., "Flexible Reverse Engineering of Web Pages with Vaquita", in Proceedings of WCRE'200: IEEE 8th Working Conference on Reverse Engineering, Stuttgart, October, 2001, IEEE Press.

[Varr02]
Varró, D., Varró, G., and Pataricza, A., *Designing the Automatic Transformation of Visual Languages*, in Science of Computer Programming, 44, 2002, pp. 205–227.

[Vand03]
Vanderdonckt, J., Limbourg, Q., Florins, M., *Deriving the Navigational Structure of a User Interface*, Proc. of 9th IFIP TC 13 Int. Conf. on Human-Computer Interaction INTERACT'2003 (Zurich, 1-5 September 2003).

W

[W3C04a]
W3C consortium, EMMA: Extensible MultiModal Annotation markup language, W3C Working Draft, 14 December 2004. Available at: http://www.w3.org/TR/emma/

[W3C04b]
W3C consortium, Voice Extensible Markup Language (VoiceXML) Version 2.0, W3C Recommendation 16 March 2004. Available at: http://www.w3.org/TR/voicexml20/

[W3C04c]
W3C consortium, XHTML+Voice Profile 1.2, 16 March 2004. Available at: http://www.voicexml.org/specs/multimodal/x+v/12/

[W3C01]
W3C consortium, XML Schema Specification, W3C Recommendation, 2 May 2001. Available at: http://www.w3.org/XML/Schema.html

X

[XIML]

XIML and Applications - A Technical Presentation, Available at: http://www.ximl.org/

ANNEX A

LABEL: due to the fact that a *label* widget does not suppose any input from the user, only three types of interaction are allowed (i.e., graphical, vocal and multimodal with redundant output):

- **Graphical interaction:**

```
<box id="b1" name="Box 1"...>
    <outputText id="OT1" name="Output 1" defaultContent="Welcome" isUnderlined="true".../>
</box>
```

- **Vocal interaction:**

```
<vocalForm id="VF1" name="Form 1"...>
    <vocalPrompt id="VP1" name="Prompt 1" defaultContent="Welcome to the UCL web site".../>
</vocalForm>
```

- **Multimodal interaction with redundant output:**

```
<vocalForm id="VF1" name="Form 1"...>
    <vocalPrompt id="VP1" name="Prompt 1" defaultContent="Welcome to the UCL web site".../>
</vocalForm>

<box id="b1" name="Box 1"...>
    <outputText id="OT1" name="Output 1" defaultContent="Welcome to the UCL web site"
isUnderlined="true".../>
    <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="Credit card".../>
    <comboBox id="CB1" name="Combo 1" isEnabled="true" currentValue="§var"...>
        <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>
```

- **Vocal interaction:**

```
<vocalMenu id="VM1" name="Menu 1" defaultContent="Select the credit card type. Choose
between"...>
    <vocalMenuItem id="VMI1" name="Item1" defaultContent="VISA"/>
    <vocalMenuItem id="VMI2" name="Item2" defaultContent="MASTERCARD"/>
    <vocalMenuItem id="VMI3" name="Item3" defaultContent="AMERICAN EXPRESS"/>
    <vocalInput id="VI1" name="Input 1" currentValue="§myMenuSelection" grammar="VISA |
MASTERCARD | AMERICAN EXPRESS".../>
    <vocalFeedback id="F1" name="Feedback 1" defaultContent="Your have selected
§myMenuSelection".../>
</vocalMenu>
```

- **Multimodal interaction with graphical assignment for input and redundant output:**

```
<vocalForm id="VF1" name="Form 1"...>
```

```
    <vocalFeedback id="F1" name="Feedback 1" defaultContent="Your have selected §var".../>
</vocalForm>

<box id="b1" name="Box 1"...>
    <outputText id="OT1" name="Output 1" defaultContent="Credit card".../>
    <comboBox id="CB1" name="Combo 1" isEnabled="true" currentValue="§var"...>
        <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="IC3" name="speaker_icon" defaultContent="speaker.jpg".../>
</box>
```

- **Multimodal interaction with vocal assignment for input and redundant output:**

```
<vocalMenu id="VM1" name="Menu 1" defaultContent="Select the credit card type. Choose
between"...>
    <vocalMenuItem id="VMI1" name="Item1" defaultContent="VISA"/>
    <vocalMenuItem id="VMI2" name="Item2" defaultContent="MASTERCARD"/>
    <vocalMenuItem id="VMI3" name="Item3" defaultContent="AMERICAN EXPRESS"/>
    <vocalInput id="VI1" name="Input 1" currentValue="§myMenuSelection" grammar="VISA |
MASTERCARD | AMERICAN EXPRESS".../>
    <vocalFeedback id="F1" name="Feedback 1" defaultContent="Your have selected
§myMenuSelection".../>
</vocalMenu>

<box id="b1" name="Box 1"...>
    <outputText id="OT1" name="Output 1" defaultContent="Credit card".../>
    <imageComponent id="IC1" name="microphone_icon" defaultContent="microphone.jpg".../>
    <comboBox id="CB1" name="Combo 1" isEnabled="false" ...>
        <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="IC2" name="speaker_icon" defaultContent="speaker.jpg".../>
</box>

<synchronization>
    <source sourceId="VI1"/>
    <target targetId="CB1"/>
</synchronization>
```

- **Multimodal interaction with equivalent input and redundant output:**

```
<vocalMenu id="VM1" name="Menu 1" defaultContent="Select the credit card type. Choose
between"...>
    <vocalMenuItem id="VMI1" name="Item1" defaultContent="VISA"/>
    <vocalMenuItem id="VMI2" name="Item2" defaultContent="MASTERCARD"/>
    <vocalMenuItem id="VMI3" name="Item3" defaultContent="AMERICAN EXPRESS"/>
    <vocalInput id="VI1" name="Input 1" currentValue="§myMenuSelection" grammar="VISA |
MASTERCARD | AMERICAN EXPRESS".../>
    <vocalFeedback id="F1" name="Feedback 1" defaultContent="Your have selected
§myMenuSelection".../>
</vocalMenu>

<box id="b1" name="Box 1"...>
    <outputText id="OT1" name="Output 1" defaultContent="Credit card".../>
    <imageComponent id="IC1" name="microphone_icon" defaultContent="microphone.jpg".../>
    <imageComponent id="IC2" name="keyboard_icon" defaultContent="keyboard.jpg".../>
    <comboBox id="CB1" name="Combo 1" isEnabled="true" currentValue="§var" ...>
        <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="IC3" name="speaker_icon" defaultContent="speaker.jpg".../>
</box>

<synchronization>
    <source sourceId="VI1"/>
    <target targetId="CB1"/>
</synchronization>
```

GROUP OF RADIO BUTTONS:

- **Graphical interaction:**

```
<groupBox id="GB1" name="Gender" currentValue="§var"...>
    <radioButton id="RB1" name="Radio 1" groupName="Gender" defaultContent="Male"
    defaultState="true" isEnabled="true" ...>
    <radioButton id="RB2" name="Radio 2" groupName="Gender" defaultContent="Female"
    defaultState="false" isEnabled="true" ...>
</groupBox>
```

- **Vocal interaction:**

```
<vocalMenu id="VM1" name="Menu 1" defaultContent="Please say your gender.Choose
between male and female"...>
    <vocalMenuItem id="VMI1" name="Item1" defaultContent="Male"/>
    <vocalMenuItem id="VMI2" name="Item2" defaultContent="Female"/>
    <vocalInput id="VI1" name="Input 1" currentValue="§myMenuSelection" grammar="Male |
    Female".../>
    <vocalFeedback id="F1" name="Feedback 1" defaultContent="Your have selected
    §myMenuSelection".../>
</vocalMenu>
```

- **Multimodal interaction with graphical assignment for input and redundant output:**

```
<vocalForm id="VF1" name="Form 1"...>
    <vocalFeedback id="F1" name="Feedback 1" defaultContent="Your have selected §var".../>
</vocalForm>

<groupBox id="GB1" name="Gender" currentValue="§var"...>
    <radioButton id="RB1" name="Radio 1" groupName="Gender" defaultContent="Male"
    defaultState="true" isEnabled="true" ...>
    <radioButton id="RB2" name="Radio 2" groupName="Gender" defaultContent="Female"
    defaultState="false" isEnabled="true" ...>
    <imageComponent id="IC3" name="speaker_icon" defaultContent="speaker.jpg".../>
</groupBox>
```

- **Multimodal interaction with vocal assignment for input and redundant output:**

```
<vocalMenu id="VM1" name="Menu 1" defaultContent="Please say your gender.Choose
between male and female"...>
    <vocalMenuItem id="VMI1" name="Item1" defaultContent="Male"/>
    <vocalMenuItem id="VMI2" name="Item2" defaultContent="Female"/>
    <vocalInput id="VI1" name="Input 1" currentValue="§myMenuSelection" grammar="Male |
    Female".../>
    <vocalFeedback id="F1" name="Feedback 1" defaultContent="Your have selected
    §myMenuSelection".../>
</vocalMenu>

<groupBox id="GB1" name="Gender" currentValue="§var"...>
    <imageComponent id="IC2" 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" ...>
    <imageComponent id="IC3" name="speaker_icon" defaultContent="speaker.jpg".../>
</groupBox>

<synchronization>
    <source sourceId="VI1"/>
    <target targetId="GB1"/>
</synchronization>
```

- **Multimodal interaction with equivalent input and redundant output:**

```
<vocalMenu id="VM1" name="Menu 1" defaultContent="Please say your gender.Choose
between male and female"...>
    <vocalMenuItem id="VMI1" name="Item1" defaultContent="Male"/>
    <vocalMenuItem id="VMI2" name="Item2" defaultContent="Female"/>
    <vocalInput id="VI1" name="Input 1" currentValue="§myMenuSelection" grammar="Male |
    Female".../>
    <vocalFeedback id="F1" name="Feedback 1" defaultContent="Your have selected
    §myMenuSelection".../>
</vocalMenu>

<groupBox id="GB1" name="Gender" currentValue="§var"...>
    <imageComponent id="IC1" name="microphone_icon" defaultContent="microphone.jpg".../>
    <imageComponent id="IC2" name="keyboard_icon" defaultContent="keyboard.jpg".../>
    <radioButton id="RB1" name="Radio 1" groupName="Gender" defaultContent="Male"
    defaultState="true" isEnabled="true" ...>
    <radioButton id="RB2" name="Radio 2" groupName="Gender" defaultContent="Female"
    defaultState="false" isEnabled="true" ...>
    <imageComponent id="IC3" name="speaker_icon" defaultContent="speaker.jpg".../>
</groupBox>

<synchronization>
    <source sourceId="VI1"/>
    <target targetId="GB1"/>
</synchronization>
```

GROUP OF CHECK BOXES:

- **Graphical interaction:**

```
<groupBox id="GB1" name="Hobbies" currentValue="§var"...>
    <checkBox id="CB1" name="Check 1" groupName="Hobbies" defaultContent="Sports"
    defaultState="true" isEnabled="true" ...>
    <checkBox id="CB2" name="Check 2" groupName="Hobbies" defaultContent="Travel"
    defaultState="false" isEnabled="true" ...>
    <checkBox id="CB3" name="Check 3" groupName="Hobbies" defaultContent="Music"
    defaultState="true" isEnabled="true" ...>
    <checkBox id="CB4" name="Check 4" groupName="Hobbies" defaultContent="Movies"
    defaultState="true" isEnabled="true" ...>
</groupBox>
```

- **Vocal interaction:**

```
<vocalMenu id="VM1" name="Menu 1" defaultContent="Please select your hobbies. Choose one
or more of the following: sport, travel, music, movies"...>
    <vocalMenuItem id="VMI1" name="Item1" defaultContent="Sports"/>
    <vocalMenuItem id="VMI2" name="Item2" defaultContent="Travel"/>
    <vocalMenuItem id="VMI3" name="Item3" defaultContent="Music"/>
    <vocalMenuItem id="VMI4" name="Item4" defaultContent="Movies"/>
    <vocalInput id="VI1" name="Input 1" currentValue="§myMenuSelection" grammar="Sports |
    Travel | Music | Movies".../>
    <vocalFeedback id="F1" name="Feedback 1" defaultContent="Your have selected
    §myMenuSelection".../>
</vocalMenu>
```

- **Multimodal interaction with graphical assignment for input and redundant output:**

```
<vocalForm id="VF1" name="Form 1"...>
    <vocalFeedback id="F1" name="Feedback 1" defaultContent="Your have selected §var".../>
</vocalForm>

<groupBox id="GB1" name="Hobbies" currentValue="§var"...>
    <checkBox id="CB1" name="Check 1" groupName="Hobbies" defaultContent="Sports"
    defaultState="true" isEnabled="true" ...>
    <checkBox id="CB2" name="Check 2" groupName="Hobbies" defaultContent="Travel"
    defaultState="false" isEnabled="true" ...>
    <checkBox id="CB3" name="Check 3" groupName="Hobbies" defaultContent="Music"
    defaultState="true" isEnabled="true" ...>
    <checkBox id="CB4" name="Check 4" groupName="Hobbies" defaultContent="Movies"
defaultState="true" isEnabled="true" ...>
    <imageComponent id="IC3" name="speaker_icon" defaultContent="speaker.jpg".../>
</groupBox>
```

- **Multimodal interaction with vocal assignment for input and redundant output:**

```
<vocalMenu id="VM1" name="Menu 1" defaultContent=" Please select your hobbies. Choose one
or more of the following: sport, travel, music, movies "...>
    <vocalMenuItem id="VMI1" name="Item1" defaultContent="Sports"/>
    <vocalMenuItem id="VMI2" name="Item2" defaultContent="Travel"/>
    <vocalMenuItem id="VMI3" name="Item3" defaultContent="Music"/>
    <vocalMenuItem id="VMI4" name="Item4" defaultContent="Movies"/>
    <vocalInput id="VI1" name="Input 1" currentValue="§myMenuSelection" grammar="Sports |
Travel | Music | Movies".../>
    <vocalFeedback id="F1" name="Feedback 1" defaultContent="Your have selected
§myMenuSelection".../>
</vocalMenu>

<groupBox id="GB1" name="Hobbies" currentValue="§var"...>
    <imageComponent id="IC1" name="microphone_icon" defaultContent="microphone.jpg".../>
    <checkBox id="CB1" name="Check 1" groupName="Hobbies" defaultContent="Sports"
    defaultState="true" isEnabled="false" ...>
    <checkBox id="CB2" name="Check 2" groupName="Hobbies" defaultContent="Travel"
    defaultState="false" isEnabled="false" ...>
    <checkBox id="CB3" name="Check 3" groupName="Hobbies" defaultContent="Music"
    defaultState="true" isEnabled="false" ...>
    <checkBox id="CB4" name="Check 4" groupName="Hobbies" defaultContent="Movies"
    defaultState="true" isEnabled="false" ...>
    <imageComponent id="IC3" name="speaker_icon" defaultContent="speaker.jpg".../>
</groupBox>

<synchronization>
    <source sourceId="VI1"/>
    <target targetId="GB1"/>
</synchronization>
```

- **Multimodal interaction with equivalent input and redundant output:**

```
<vocalMenu id="VM1" name="Menu 1" defaultContent=" Please select your hobbies. Choose one
or more of the following: sport, travel, music, movies "...>
    <vocalMenuItem id="VMI1" name="Item1" defaultContent="Sports"/>
    <vocalMenuItem id="VMI2" name="Item2" defaultContent="Travel"/>
    <vocalMenuItem id="VMI3" name="Item3" defaultContent="Music"/>
    <vocalMenuItem id="VMI4" name="Item4" defaultContent="Movies"/>
    <vocalInput id="VI1" name="Input 1" currentValue="§myMenuSelection" grammar="Sports |
    Travel | Music | Movies".../>
    <vocalFeedback id="F1" name="Feedback 1" defaultContent="Your have selected
    §myMenuSelection".../>
</vocalMenu>

<groupBox id="GB1" name="Hobbies" currentValue="§var"...>
    <imageComponent id="IC1" name="microphone_icon" defaultContent="microphone.jpg".../>
    <imageComponent id="IC2" name="keyboard_icon" defaultContent="keyboard.jpg".../>
    <checkBox id="CB1" name="Check 1" groupName="Hobbies" defaultContent="Sports"
    defaultState="true" isEnabled="true" ...>
    <checkBox id="CB2" name="Check 2" groupName="Hobbies" defaultContent="Travel"
    defaultState="false" isEnabled="true" ...>
    <checkBox id="CB3" name="Check 3" groupName="Hobbies" defaultContent="Music"
    defaultState="true" isEnabled="true" ...>
    <checkBox id="CB4" name="Check 4" groupName="Hobbies" defaultContent="Movies"
    defaultState="true" isEnabled="true" ...>
    <imageComponent id="IC3" name="speaker_icon" defaultContent="speaker.jpg".../>
</groupBox>

<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".../>
    <listBox id="LB1" name="List 1" isEnabled="true" currentValue="§var" ...>
        <item id="IT1" name="Item 1" defaultContent="Chris Hay".../>
        <item id="IT2" name="Item 2" defaultContent="Lee Hardy".../>
    </listBox>
</box>
```

- **Vocal interaction:**

```
<vocalMenu id="VM1" name="Menu 1" defaultContent="Please select your favorite singers: Chris
Hay, Lee Hardy,..."...>
    <vocalMenuItem id="VMI1" name="Item1" defaultContent="Chris Hay"/>
    <vocalMenuItem id="VMI2" name="Item2" defaultContent="Lee Hardy"/>
    <vocalInput id="VI1" name="Input 1" currentValue="§myMenuSelection" grammar="Chris Hay
    | Lee Hardy".../>
    <vocalFeedback id="F1" name="Feedback 1" defaultContent="Your have selected
    §myMenuSelection".../>
</vocalMenu>
```

- **Multimodal interaction with graphical assignment for input and redundant output:**

```
<vocalForm id="VF1" name="Form 1"...>
    <vocalFeedback id="F1" name="Feedback 1" defaultContent="Your have selected §var".../>
</vocalMenu>

<box id="b1" name="Box 1"...>
    <outputText id="OT1" name="Output 1" defaultContent="Singers".../>
    <listBox id="LB1" name="List 1" isEnabled="true" currentValue="§var" ...>
        <item id="IT1" name="Item 1" defaultContent="Chris Hay".../>
        <item id="IT2" name="Item 2" defaultContent="Lee Hardy".../>
    </listBox>
    <imageComponent id="IC3" name="speaker_icon" defaultContent="speaker.jpg".../>
</box>
```

- **Multimodal interaction with vocal assignment for input and redundant output:**

```
<vocalMenu id="VM1" name="Menu 1" defaultContent="Please select your favorite singers: Chris
Hay, Lee Hardy,..."...>
    <vocalMenuItem id="VMI1" name="Item1" defaultContent="Chris Hay"/>
    <vocalMenuItem id="VMI2" name="Item2" defaultContent="Lee Hardy"/>
    <vocalInput id="VI1" name="Input 1" currentValue="§myMenuSelection" grammar="Chris Hay
    | Lee Hardy".../>
    <vocalFeedback id="F1" name="Feedback 1" defaultContent="Your have selected
    §myMenuSelection".../>
</vocalMenu>

<box id="b1" name="Box 1"...>
    <outputText id="OT1" name="Output 1" defaultContent="Singers".../>
    <imageComponent id="IC1" name="microphone_icon" defaultContent="microphone.jpg".../>
    <listBox id="LB1" name="List 1" isEnabled="false" currentValue="§var" ...>
        <item id="IT1" name="Item 1" defaultContent="Chris Hay".../>
        <item id="IT2" name="Item 2" defaultContent="Lee Hardy".../>
    </listBox>
    <imageComponent id="IC3" name="speaker_icon" defaultContent="speaker.jpg".../>
</box>

<synchronization>
    <source sourceId="VI1"/>
    <target targetId="LB1"/>
</synchronization>
```

- **Multimodal interaction with equivalent input and redundant output:**

```
<vocalMenu id="VM1" name="Menu 1" defaultContent="Please select your favorite singers: Chris
Hay, Lee Hardy,..."...>
    <vocalMenuItem id="VMI1" name="Item1" defaultContent="Chris Hay"/>
    <vocalMenuItem id="VMI2" name="Item2" defaultContent="Lee Hardy"/>
    <vocalInput id="VI1" name="Input 1" currentValue="§myMenuSelection" grammar="Chris Hay
    | Lee Hardy".../>
    <vocalFeedback id="F1" name="Feedback 1" defaultContent="Your have selected
    §myMenuSelection".../>
</vocalMenu>

<box id="b1" name="Box 1"...>
    <outputText id="OT1" name="Output 1" defaultContent="Singers".../>
    <imageComponent id="IC1" name="microphone_icon" defaultContent="microphone.jpg".../>
    <imageComponent id="IC2" name="keyboard_icon" defaultContent="keyboard.jpg".../>
    <listBox id="LB1" name="List 1" isEnabled="true" currentValue="§var" ...>
        <item id="IT1" name="Item 1" defaultContent="Chris Hay".../>
        <item id="IT2" name="Item 2" defaultContent="Lee Hardy".../>
    </listBox>
    <imageComponent id="IC3" name="speaker_icon" defaultContent="speaker.jpg".../>
</box>

<synchronization>
    <source sourceId="VI1"/>
    <target targetId="LB1"/>
</synchronization>
```

ANNEX B

NAC LHS RHS

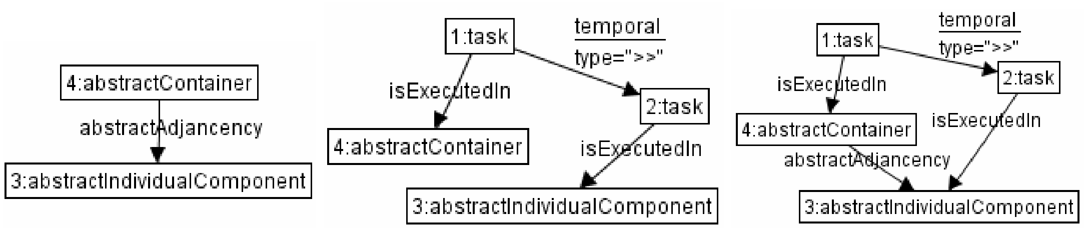


Figure B-1. Creating abstract adjacency for <AC, AIC> couple

NAC LHS RHS

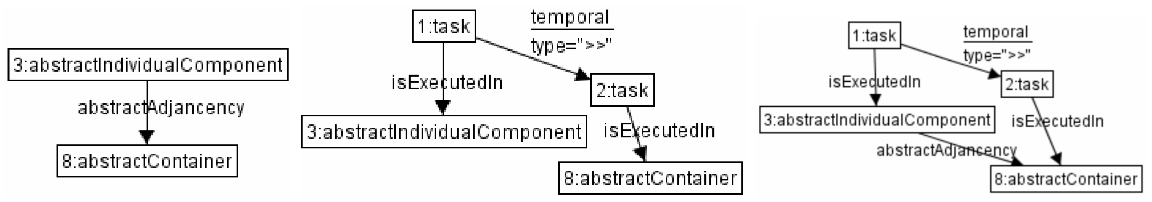


Figure B-2. Creating abstract adjacency for <AIC, AC> couple

NAC LHS RHS

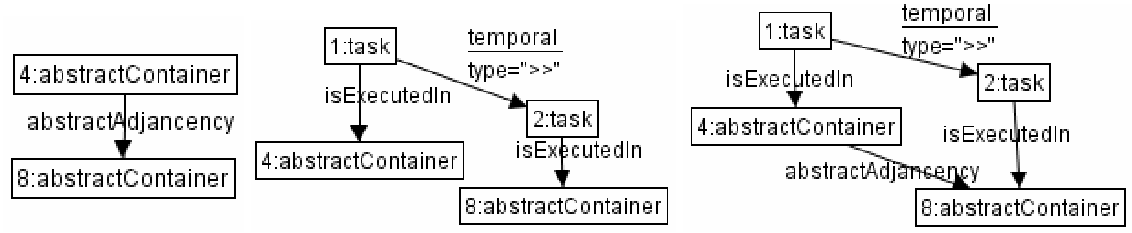


Figure B-3. Creating abstract adjacency for <AC, AC> couple

NAC LHS RHS

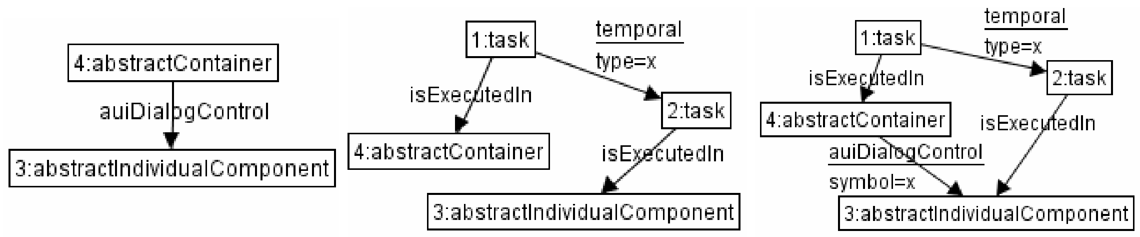


Figure B-4. Deriving Abstract Dialog Control for <AC, AIC> couple

NAC LHS RHS

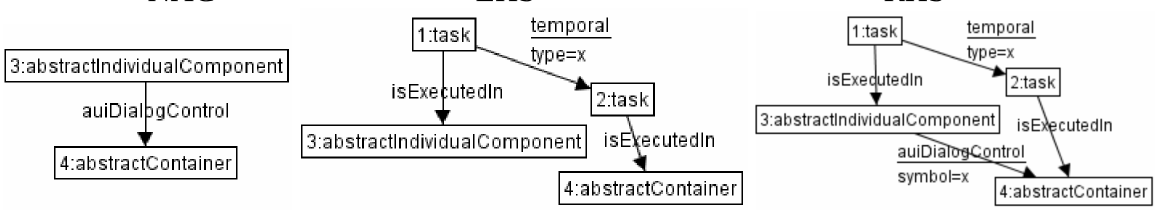


Figure B-5. Deriving Abstract Dialog Control for <AIC, AC> couple

NAC LHS RHS

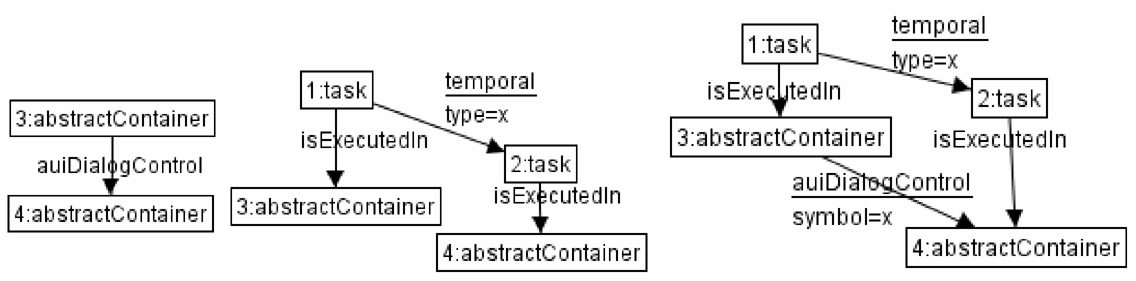


Figure B-6. Deriving Abstract Dialog Control for <AC, AC> couple

NAC LHS RHS

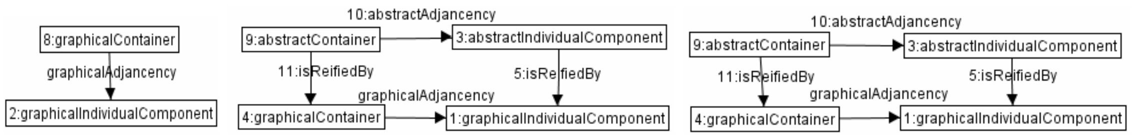


Figure B-7. Generation of Graphical Adjacency relationships for <GC, GIC> couples

NAC LHS RHS

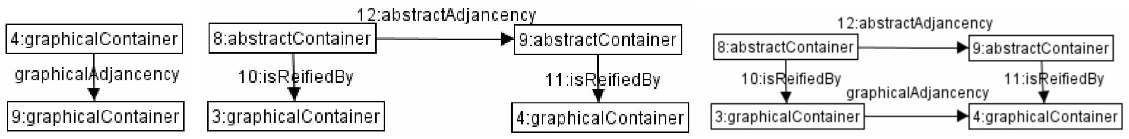


Figure B-8. Generation of Graphical Adjacency relationships for <GC, GC> couples

NAC LHS RHS

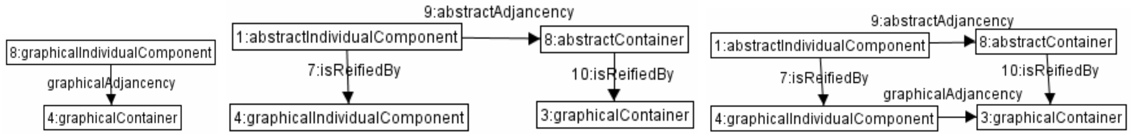


Figure B-9. Generation of Graphical Adjacency relationships for <GIC, GC> couples

NAC LHS RHS

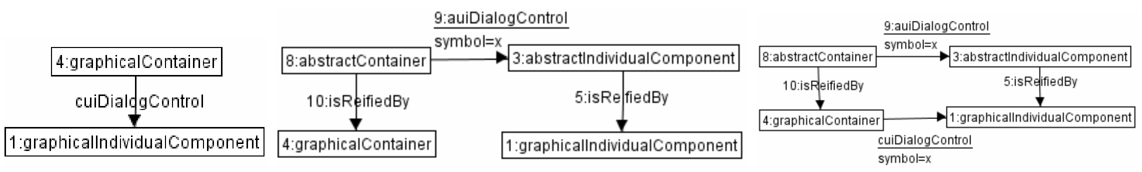


Figure B-10. Generation of Concrete Dialog Control relationships for <GC, GIC> couples

NAC LHS RHS

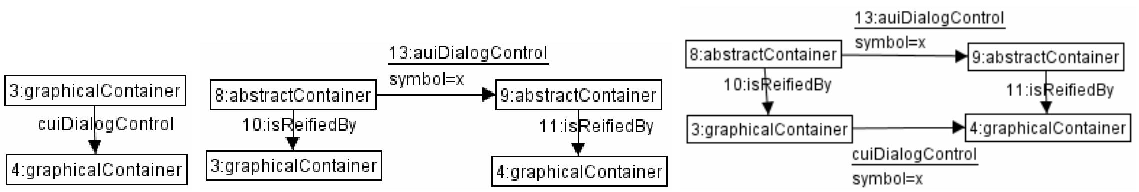


Figure B-11. Generation of Concrete Dialog Control relationships for <GC, GC> couples

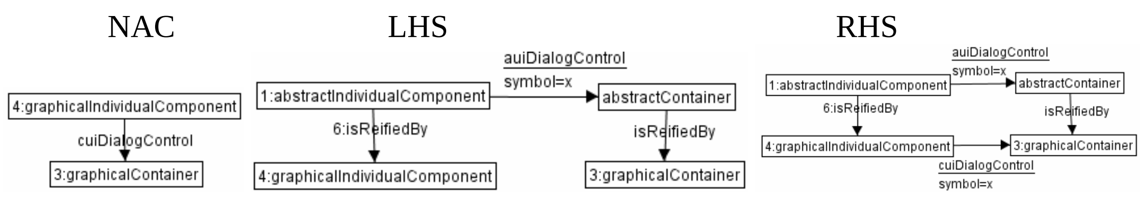


Figure B-12. Generation of Concrete Dialog Control relationships for <GIC, GC> couples

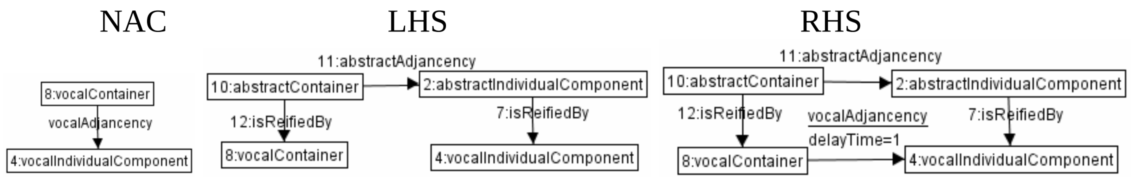


Figure B-13. Generation of Vocal Adjacency relationships for <VC, VIC> couples

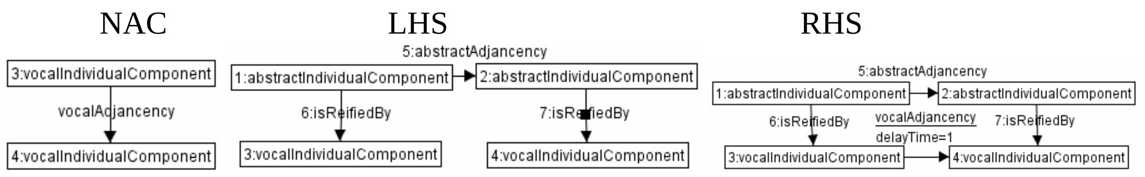

NAC
LHS
RHS
3:vocalIndividualComponent
vocalAdjancency
4:vocalIndividualComponent
1:abstractIndividualComponent
5:abstractAdjancency
2:abstractIndividualComponent
6:isReifiedBy
7:isReifiedBy
3:vocalIndividualComponent
4:vocalIndividualComponent
1:abstractIndividualComponent
5:abstractAdjancency
2:abstractIndividualComponent
6:isReifiedBy
vocalAdjancency
delayTime=1
7:isReifiedBy
3:vocalIndividualComponent
4:vocalIndividualComponent

Figure B-14. Generation of Vocal Adjacency relationships for <VIC, VIC> couples

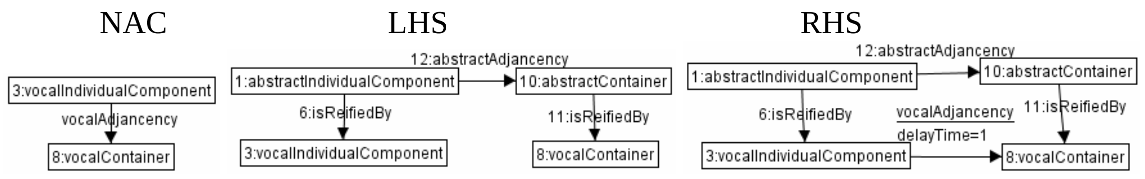


Figure B-15. Generation of Vocal Adjacency relationships for <VIC, VC> couples

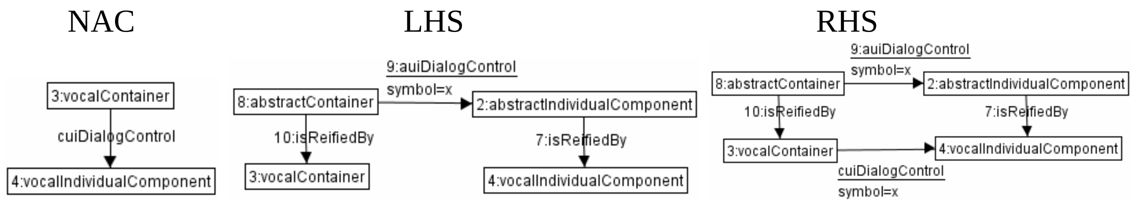


Figure B-16. Generation of Concrete Dialog Control relationships for <VC, VIC> couples

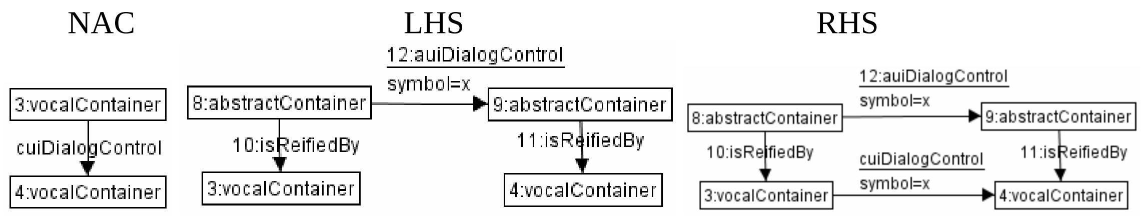


Figure B-17. Generation of Concrete Dialog Control relationships for <VC, VC> couples

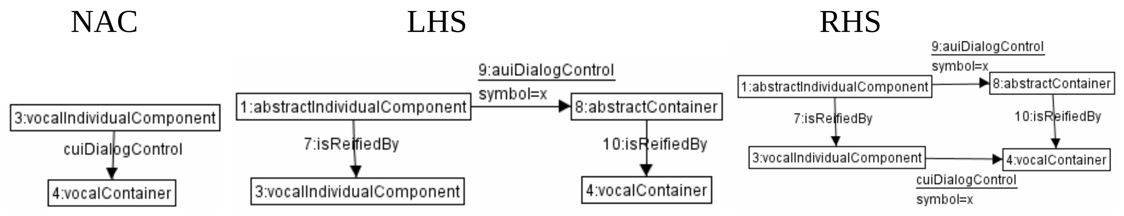


Figure B-18. Generation of Concrete Dialog Control relationships for <VIC, VC> couples