

## Towards Canonical Task Types for User Interface Design

Juan Manuel González-Calleros  
Josefina Guerrero-García  
Jean Vanderdonckt  
Université catholique de Louvain  
Place des Doyens, 1 – B-1348  
Louvain-la-Neuve, Belgium {josefina.guerrero,  
juan.m.gonzalez, jean.vanderdonckt}@uclouvain.be

Jaime Muñoz-Arteaga  
Sistemas de Información  
Universidad Autónoma de Aguascalientes  
Av. Universidad No. 940, Col. Bosques, 20100  
Aguascalientes, Aguascalientes (México)  
jmunozar@correo.uaa.mx  
CENIDET, Internado Palmira S/N, Col. Palmira, C.P.  
62490, Cuernavaca,  
Morelos. México

**Abstract**— Task models are the cornerstone of user-centred design methodologies for user interface design. Therefore, they deserve attention in order to produce them effectively and efficiently, while guaranteeing the reproducibility of a task model: different persons should in principle obtain the same task model, or a similar one, for the same problem. In order to provide user interface designers with some guidance for task modelling, a list of canonical task types is proposed that offers a unified definition of frequently used tasks types in a consistent way. Each task type consists of a task action coupled with a task object, each of them being written according to design guidelines. This list provides the following benefits: tasks are modelled in a more consistent way, their definition is more communicable and shared, task models can be efficiently used for model-driven engineering of user interfaces.

**Keywords:** User interfaces, User Interface Description Language, task types, task model, Model-Based User Interface development.

### I. INTRODUCTION

The task model is today a cornerstone of many activities carried out during the User Interface (UI) development life cycle, such as, but not limited to: user-centred design, task analysis and task modelling, model-driven engineering of user interfaces, human activity analysis, safety critical systems, and real-time systems. Modelling a task based on well-defined semantics and using a well-understood notation are key aspects, but the many degrees of freedom offered by task modelling should not let us to forget the quality of the resulting task model. Over time, we observed the following forms of laxism:

- *Incompleteness*: labels, definitions, goals, and properties used for a task suffer from many drawbacks such as short name, name without action verb or without object (and therefore non-compliant with the traditional interaction paradigm of action+object), name that is incompatible with its definition, no usage of standard classification.

- *Inconsistency*: labels, definitions, goals, and properties used for a task do not have unique names (e.g., a label, a goal is duplicated), there are some homonyms; there are some synonyms (e.g., tasks having the same semantics but wearing different names).
- *Incorrectness*: labels, definitions, goals, and properties used for a task violate some of Meyer's seven sins of specification (i.e., noise, silence, surspecification, contradiction, ambiguity, forward reference, and suroptimism).

Not only those forms of laxism are observed during the activity of task modelling itself, but they are then propagated, if not amplified, in the rest of the UI development life cycle since this rest is effectively based on the task model. The damages are even more important until they reach the stage of the final UI.

In this paper a list of canonical task types is proposed that addresses the aforementioned concerns of task modelling. With this list our goal is to provide methodological means to systematically derive UI. For that purpose the list provides information that can be used for concretization of the task in a UI, for instance, widget selection or dialog specification. The list is just about the name of the task and not its structure, thus remaining flexible for task modelling. In practice, this work is illustrated in a real life case study.

### II. STATE OF THE ART

The UI interaction is composed of two elements: (1) the task type, in the literature sometimes referred as UI action or activity; and (2) the task item that is manipulated or required in the UI interaction [4]. Both attributes are relevant to design interactive systems using task models. In HCI several works [6][17][18][20] rely on task to derive UIs. However, most of their decision for UI generation is more intuitive rather than systematic. When heuristics are used for the selection, researchers based this decision on specific attributes of the task model, such as the task type [17], [20].

Table 1: List of Canonical User Interface Action task types

Action Type	Task name	Definition	Examples
<b>Convey</b>	Communicate, Transmit, Call, Acknowledge, Respond, Answer, Suggest, Direct, Instruct, Request	The action to exchange information	Show details Switch to summary
<b>Create</b>	Input, Encode, Enter, Associate, Name, Introduce, Insert, Assemble, Aggregate, Add	Specifies the creation of an item instance	New customer, blank slide
<b>Delete</b>	Eliminate, Remove, Cut, Ungroup, Disassociate	The action of deleting an item	Break connection, Delete file/slide
<b>Duplicate</b>	Copy	Specifies the copy of an item	copy address, duplicate slide
<b>Filter</b>	Segregate, Set aside	The action of filtering an item	Filter email, segregate any modification on a data base when backing up
<b>Mediate</b>	Analyze, Synthesize, Compare, Evaluate, Decide	The action of intercede task items	Compare products characteristics on a online store
<b>Modify</b>	Change, Alter, Transform, Tuning, Rename, Segregate, Resize, Collapse, Expand	An action of modifying an item	Change shipping address, Tuning volume
<b>Move</b>	Relocate, Hide, Show, Position, Orient, Path, Travel	the action to change the location of an item	Put into address list, move up/ down?
<b>Navigation</b>	Go to	the action to find the way through containers	Navigation bar on a web browser
<b>Perceive</b>	Acquire, Detect, Search, Scan, Extract, Identify, Discriminate, Recognize, Locate, Examine, Monitor, Scan, Detect,	The action of identifying items and/or information from the items	Locate a destination in a map, observe the status bar while installing
<b>Reinitialize</b>	Wipe out, Clear, Erase	The action of cleaning an item	Clear form,
<b>Select</b>	Pick, Choose	selection between items	group member picker, object selector
<b>Trigger</b>	Initiate/Start, Play, Search, Active, Execute, Function, Record, Purchase	Specifies the beginning of an operation	Play audio/video file
<b>Stop</b>	End, Finish, Exit, Suspend, Complete, Terminate, Cancel	Specifies the end of an action	Stop searching/playing, cancel register
<b>Toggle</b>	Activate, Deactivate, Switch	The existence of two different states of an item	Bold on/off, encrypted mode,

Naming task types using a restricted set of names has been proposed for different application domains: GUI [4], web interaction [12], input devices [9], multimodal interaction [1][2][11] [15]. Still the names are dependent on the interaction technique to be used and are not generic enough and independent of the implementation (a characteristic that must be accomplished by definition in task modelling). Not everything is lost and few attempts propose canonical description of task types [5][13], however, they suffer from not being wide enough in order to cope a more general set of task types rather than being too concrete with a limited set of values.

Lenorovitz et al. on his review of human computer interaction ended with a list of frequent tasks interactive [14] separated the task categories into: user interactive actions, user actions and system actions. In this review user actions are more related to cognitive issues. User interactive actions correspond to the tangible manipulation of the system. System actions normally are transparent to the user, they user do not know what is happening at the system level. Constantine proposes a list of canonical Action types and action items, enabling a refined expression of the nature of leaf tasks (sometimes called action tasks or leaf tasks) [4]. This expression qualifies a UI in terms of abstract actions it supports. The list is twofold: a verb describes the type of activity at hand; an expression designates the type of object on which the action is operated. By combining these two dimensions a derivation of interaction objects supposed to support a task becomes possible.

There is a lot of work on HCI patterns ([www.hcipatterns.org](http://www.hcipatterns.org)). More particularly, patterns for task modelling. Even that the pattern-based approach seems interesting it is out of the scope of this paper to review HCI patterns literature exhaustively. There is no need to do that as patterns are contradictory to model-driven (MD) paradigm that is followed in the context of this paper. While MD methods are structured is a common way, patterns are poorly structured or are structured in different ways, thus, leading to inconsistency when relying on them. Of course, the use of our work can lead to task patterns, similarly to those presented in [7], but we are not proposing a set of patterns.

### III. CANONICAL TASK TYPES

Looking at the previous work, from which more than two hundred names were identified, and based on task models found in the literature and done in HCI courses, an agreement was found that: it is not feasible to include all possible names for the task type. Something is needed to reduce the name-space if further transformations are expected from this attribute. From the literature [2][8][10][16][17][18][19] it is clear that the task type is used in further transformations to generate UIs from task models. Then a list of canonical task types is needed.

Based on existing list of action types [1][2][4][5][9][11][12][13][14][15], a comparison of names, meaning and domain applicability was done where common task types were grouped (second column in Table 1). While looking at the examples and the definition it was clear that they belong to the same category. From there the most abstract or modality and platform independent name was chosen.

In next section the applicability of the action types is shown along with a method to develop UIs from task models [9]. Notice that this set of task types is just a recommendation to be followed but designers could chose any other name that fits better its own purpose.

#### IV. APPLICABILITY OF TASK TYPES

In this section a set of guidelines to write task models is proposed. That is later used in the context of a Model-Based development method for UI development.

##### A. Task name

Our empirical experience, from courses and literature review, reveals that authors normally end in similar structures of the task model with minimal variations. However, the way authors name tasks varies considerably. While authors can select tasks' names without following any basis, automatic transformation of task models to User Interfaces becomes almost impossible considering the infinite variety of names that authors can choose for the task. From Table 1 the list we propose a set of names (the task type, synonyms and sub-types) for naming tasks. Ideally a task should be named with two elements: task type and the object they manipulate, see Figure 1, and authors should use for the task type any action from the list. Even, with the names are not considered correct or representative to the task, authors will not be force to use this methodological guidance to name the tasks, they might add new names. Our goal is to at least keep homogeneity for the set of action types.

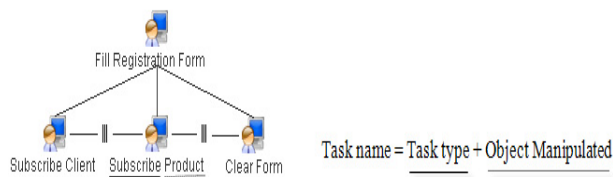


Figure 1 Guideline for task naming

##### B. Task models attributes: task type and task item

A UI action type is the result of a combination of *Action types* and action items. So far, action types have been discussed in the canonical list proposed. Once the author name the task using the previous guideline the next step is to set two attributes of the task: the action type and action item (Figure 2). The action type can be automatically assigned from the task type if authors chose any abstract value (first column in Table 1) of the action types. When it

is not the case but an optional name (second column in Table 1) is used instead, the corresponding action type for that synonym must be assigned. Regarding the task item, the decision of which value must be assigned is based on the value the task manipulates. In this case we refer to the domain model of the problem and we have to look the class(es) the task manipulates, so as the variables and methods it manipulates. The assignation of this parameter is as follow:

- Operation if the task manipulates a method.
- Element specifies that the item has a single characteristic, normally associated to a variable in a class, for instance the name of a person.
- Container specifies that the item is an aggregation of elements, normally variables member of the same class. For instance, the attributes describing a book might be contained together.
- Collection of elements specifies that the item is composed of a list of elements or containers. For instance, customer registration container (name, address) and the shopping list containing: items descriptions.

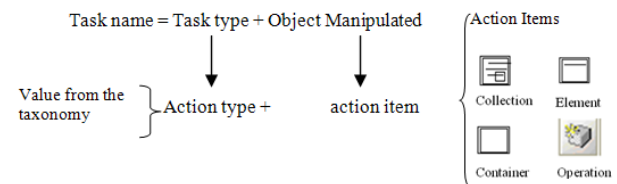


Figure 2 Guideline for task type naming





##### C. Combining User Interface Actions Types with User Categories

The combination of task categories and UI Action types (Action type and action item) provides extra information for UI generation. To understand the mean of the combinations between task item and user categories, an analysis was conducted for each abstract task type included in Table 1. As an example of the way task types have been investigated, Table 2 and Table 3 illustrate tasks categories for reinitializing, selecting and mediate. For each task there is a different meaning while being combined with different user's categories so as the task item they manipulate. The combination of these elements is useful for further concretization of the task.

The reinitialize task refers to an item that either erases or cleans certain fields (text field in the graphical modality) in a UI. This is the result on the visual part but this action might have impact as well at the data level, section E provides some hints for the dialog generation. At the data level it implies to restore the default value. Reinitialize an element, a collection or a container on a UI represents almost the same. However, the executor of the task, the user interacting with the system, the system or the cognitive decision making of the user, has different interpretations. While system and user categories might

have, in principle, no representation in the UI. When the task is interactive, the user might need an explicit mechanism to execute the reinitialize task item, i.e. an abstract interaction object (AIO), concept introduced in [2] will be concretized in further steps. In some cases the reinitialize task is implicit to the nature some other task types, for instance, in a mailing website creating a new user account normally implies the use of a form where users fill a set of fields with personal data. Each element on the UI, unless something else is predefined, can be reinitialized by the user without the use of a reinitialize task for each element. It is always possible to erase an entry in a form and this does not mean that for each entry there will be a need for a supplementary task to specify that it can be reinitialized. This is something that at the implementation level is assumed.

Table 2 Reinitialize User Interface actions examples





Task Type	Task Item	User category		
		Interactive	System	User
Reinitialize		All the customer registration elements (name, address) are set to their default values	A system response to restore a task item to its default value.	Make the decision of reinitializing a task item
		A button that clear a form or restore to the default values		
		Erasing a text field		
		Pressing a button to restore a variable to its default value		

D. User Interface Concretization of the task

Designers can benefit from the list as the design space of UI actions is reduced to a set that is easier to handle. During a second phase of the process of developing UIs the task action can be mapped to a correspondent UI. The method proposed relies on the User Interface Description Language (UIDL) UsiXML [15]. Composed of the following models: *Task Model* that represents user’s tasks along with their logical and temporal ordering; *Domain Model* with concepts as classes, attributes, methods, objects and domain relationships; *Abstract User Interface Model (AUI)* that represents a canonical expression of the rendering and manipulation of the domain concepts and functions in a way that is as independent as possible from modalities and computing platform specificities; *Concrete User Interface Model (CUI)*, a UI model allowing a specification of an appearance and behavior of a UI with elements that can be perceived by users; *Context Model*, a model describing the three aspects of a context of use in

which an end user is carrying out an interactive task with a specific computing platform in a given surrounding environment; *Inter-Model Relationships* (i.e. mapping model), model integration is a well-known issue in transformation driven development of UI; *Final User Interface (FUI)* that Corresponds to the code generation for common languages such as: Java, Flash, HTML or even for Three-dimensional UIs [8]. As in the task model, the AUI model uses the same attributes to specify the UI action: action type and task item. In addition, the abstract level incorporates the facet concept.

Table 3 Mediate User Interface actions examples

Task Type	Task Item	User category		
		Interactive	System	User
Mediate		Compare products by price	Google search evaluating the best ranked pages to present the results of a query.	Analyze the data details (author, name, publisher, ...) of a book
		Compare side by side documents in word	Decide the layout of a slide when creating a new one	Compare a list of books
		Evaluate a video watched on YouTube	Evaluate the security risk of a password	Determine the date of a trip
		Decide which operation to apply to a combination of CTRL keys.	Propose different arrangement of the results of a query.	Decide which operation will be used with a special key on a joystick

The action Type attribute of a facet enables the specification of the type of action an *Abstract Individual Component (AIC)* allows to perform. The action Item attribute characterizes the item that is manipulated by the AIC. The AUI Model as well as the Task Model is independent of any modality of interaction. The set of possible AUI facets are:

- An input facet describes the input action supported by an AIC.
- An output facet describes what data may be presented to the user by an AIC.
- A navigation facet describes the possible container transition, a particular AIC may enable.
- A control facet describes the links between an AIC and system functions, i.e., methods from the domain model when existing.

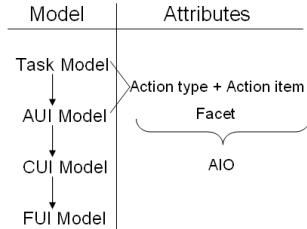


Figure 3 Meta-model of the AUI Model

Unfortunately, it is not enough while the action type and action item combined with the facet to properly select the AIO. An example can be used to clarify this situation. Assuming that the user interface action type corresponds to a select of a collection of elements, then, several are the potential AIOs that can be used such as: combo box, radio button group, text fields, etc. The problem became then on deciding the appropriate AIO depending on the context of use, the type of value to be selected, and the domain. For that purpose, the rest of UsiXML meta-models can be used. While models already exist and have been discussed in other papers [8] [10] [15] [19] [20], the scope of this paper is on how to use them and to provide guidelines on the proper selection of AIOs.

Table 4 Simple selection interactive objects. Source [2]

Number of values	Known Domain	Mixed Domain	Unknown domain
[2, 3]	<input type="radio"/> Value 1 <input type="radio"/> Value 2	<input type="radio"/> Value 1 <input type="radio"/> Value 2 Value <input type="text"/>	
[4, 7]	GroupBox <input type="radio"/> Value 1 <input type="radio"/> Value 2 <input type="radio"/> Value 3 <input type="radio"/> Value 4	GroupBox <input type="radio"/> Value 1 <input type="radio"/> Value 2 <input type="radio"/> Value 3 <input type="radio"/> Value 4 <input checked="" type="radio"/> Other <input type="text"/>	Value <input type="text"/>
[8, 50]	Value 1 Value 2 Value 3 Value 4 Value 5 Value 6 Value 7	Value 1 Value 2 Value 3 Value 4 Value 5 Value 6 Value 7	
[50, ∞]	Value 1 Value 2 Value 3 Value 4 Value 5 Value 6 Value 7	Value 1 Value 2 Value 3 Value 4 Value 5 Value 6 Value 7	

Some heuristics has been described in [16] for the mapping of abstract description of the UI to a real implementation, still, a complete set of abstract descriptors, what we called task types, is not yet available. Moreover, the possible mappings and guidelines to support the correct transfer from task model to UI widgets, as shown in [2] [13], is also relevant. Because it is not enough just to use arbitrarily a combo box for selecting a value, as Bodart and Vanderdonckt [2] proposes the concretization of the selected task must be based of the number of values to be selected, see Table 4. While this

characteristics are relevant in further transformations, following model-driven methods to derive UIs [15] [16] [17] [18] [20], describing the task types using a good characterization of user interface actions provides good basis for the concretization of the UI. For instance, the selection of a simple value can be mapped to a radio button group or a list box, the difference relies on the number of possible values to select. This characteristic could be part of the design, as used in [15] and described in detail in [19], the domain model in combination with the task model provide semantic information that can be further used on the specification of an abstract User Interface.

#### E. Some Hints to Derive the Dialog

The task reinitialize in a UI implies some other modifications such as erase or clean certain fields (text field in the graphical modality). At the data level it implies to restore the default value. In some cases the task type is implicit in some other task types, for instance, when creating a new account there are fields to fill, independently on the modality, it is always possible to erase an entry. This do not means that for each entry there will be a need for a supplementary task to specify that it can be reinitialized. This is something that at the implementation level is assumed. On the contrary, when we reinitialize a collection this has a dipper and more general impact. The key issue is to identify at the task level what concepts will be reinitialized if a task is set as reinitialize. Assuming that in the task three structures a reinitialize task is found then all the siblings of that task will be affected by the operation.

### V. CASE STUDY: THREE DIMENSIONAL UI GENERATION FOR WORKFLOW INFORMATION SYSTEMS

So far, the applicability of the canonical list of task types has been illustrated in the context of a Model-Based approach for developing UIs. In this section the feasibility of this method is shown. The problem description and the design space of the solution are captured in a workflow diagram [10] using Petri-nets graphical notation. A workflow model [10] is composed in process, which are then decomposed in tasks. Tasks are object of transformation to derive UIs, as described in section IV.D. The CUI representation of that task, as supported in UsiXML [20], could be a GUI, Three-Dimensional UI (3DUI) [8], vocal UI, and haptic. For this example the 3DUI was selected but any other solution could be obtained similarly. Notice that the transformational approach required has been reported in previous works [8] [19][20][15] and it is out of the scope of this paper.

Organizing a trip, a travel agency executes several tasks. When a customer arrives, first task is registered the customer's data in the system; then an employee searches for opportunities which are communicated to the customer. It is possible to search by different options as price, hotel or airline. Then the customer will be contacted to find out

whether she or he is still interested in the trip and whether more alternatives are desired. There are three possibilities: (1) the customer is not interested at all, (2) the customer would like to see more alternatives, and (3) the customer selects an opportunity. If the customer selects a trip, then it is booked. At the same time, one or two types of insurance are prepared if they are desired. A customer can take insurance for trip cancellation or/and for baggage loss. Note that a customer can decide not to take any insurance, just trip cancellation insurance, just baggage loss insurance, or both types of insurance. Two weeks before the start date of the trip, the documents are sent to the customer by e-mail. A trip can be cancelled at any time after completing the *make reservation* task (including the insurance) and before the start date. Notice that customers without insurance for trip cancellation can cancel the trip but will get no refund. Based on this informal description, the following models were generated:

- A process model to represent the general view of the example
- Tasks models to represent how the process is developing
- A Table 5 using task types, task items and identify the user category.

The process has a place which serves as the start condition and a place which serves as the end condition. First, the tasks Register data, Search for a trip, and Send to customer are executed sequentially. This last task has an OR-split with three possible outcomes: (1) the customer is not interested in any opportunity, (2) the customer would like to see more options, or (3) the customer selects an opportunity. Task cancel is an explicit OR-join; it can only be executed after Make reservation and before Start trip task. Now, we can divide each task in several sub-tasks with task models, so we have the total collection of task that are involve in this example.

*Register data* is composed of three tasks: *Fill out form*, *Clear form* and *Confirm data*. With *Fill out form* task it is

possible to get personal data of customer as name, zip code, gender, and age. *Clear form* task is useful if for any reason the employee decide to eliminate the data that already were typing. After typing the customer's data, the employee can verify the correct filling out of the form and send the data to a data base. *Search for a trip* task is split in *Type trip data*, *Show opportunities* and *Choose opportunities* tasks. First is necessary to introduce some references of the trip as destination city, trip dates, number of passengers, etc. Also is necessary to choose how many rooms and what kind of rooms. Then, the system will search for options considering the previous parameters. Finally, the employee could analyze the opportunities that the system found. Before select one or more options, the employee can make a new search in two ways; try to find a new option or just to change one parameter. The next task is *Send to customer* by e-mail the opportunities that are appropriate for her or him. If the customer decide to take an opportunity, then next task is *Make a reservation*. *Make a reservation* task is divided in *Book a trip* and *Offer insurance* tasks. Booking a trip needs additional passengers' information as name, age (to separate adults, children and babies), and the identification number. After send these data, the system makes the reservation and display a reservation number and the invoice, which will be send to the customer. As we can see on the description of the example, the customer can take insurance for trip cancellation or for baggage loss, not to take any insurance, or both types of insurance. After *Make reservation*, the next task is the payment. *Pay invoice* task is out of this example because the tasks that need to be developed to pay could be different for each customer. Once customer pays, the system notify to the employee in order that she or he can confirm the trip and send e-tickets to the customer. Customer can cancel the trip after she or he made the reservation and until the trip starts. The employee needs to verify the reservation conditions to know if the customer buys some insurance or not.

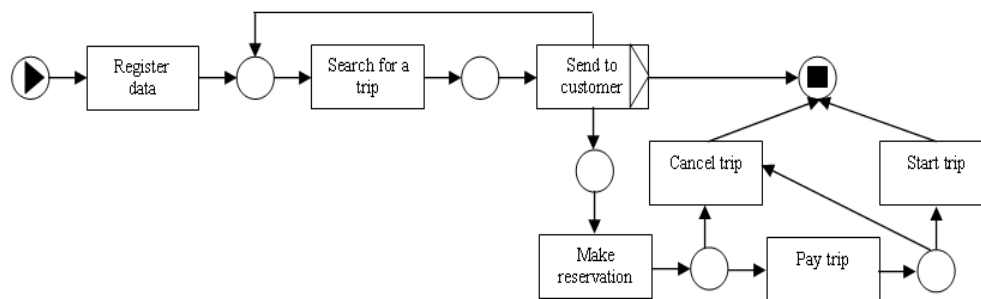


Figure 4 Organize a trip Process mode

For each task in the process a task model details the steps needed to accomplish it. In Table 5, some sub-tasks from task models are listed with the purpose to show the use of task types, combined with task items and user categories. First, the user interacts with the system (Interactive-user category) to *type its user data*. This task includes the manipulation of the user data (name, zip code,

gender, age) one by one (element-task item). A couple of cognitive tasks (*decide to remove data* and *Check data*) which indicate for our system to be aware of the feedback provided by the system during the interaction. If after *deciding to remove data* the user performs the action *clear data* then this interactive action should be reflected as a service available in the UI and associated to a function

corresponding to the clear form behaviour. The rest of the table is part of other task models. Notice the combination of the three attributes (task type, task action and user category) as later on this information is relevant for the concretization of the task in a UI.

Table 5 Excerpt from the task types categorization for the organize a trip case study

Task name	Task Type	Task Item	User category
Type user data	Create	Element	Interactive
Decide to remove data	Delete	Collection	User
Clear data	Delete	Collection	Interactive
Check data	Mediate	Collection	User
Send data	Communicate	Collection	Interactive
Store data	Create	Collection	System
Search options	Trigger	Element	System
Display options	Communicate	Collection	System
Analyze opportunities	Mediate	Collection	User
Display information	Communicate	Collection	System
Cancel	Stop	Element	Interactive

The concretization for *type user data* task is shown in Figure 5. For instance, in Table 6 the list of possible concretizations for the task *type user* is described.

Table 6 Abstract Interaction Component selection from the User Interface Action types

User Interface Action Types	Facet Specification	Information to take into account	Possible Abstract Interaction Component
“create name” and “create zip Code”	Create attribute value	Data type, domain characteristics	A text output with a text input associated to it
“select gender and select age Category”	Select attribute value + selection values known	Data type, domain characteristics, selection values	A dropdown list, a group of radio buttons textual or characters.

The next step is the appropriate selection of graphical elements. The selection of graphical elements is based on ergonomic rules and guidelines. While for creating name and zip code there is no other option than text inputs and outputs. For selecting the category and the gender the option is a radio button accordingly to the guideline described in Table 4.

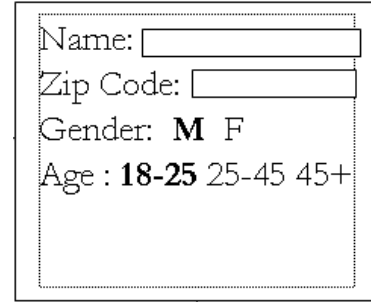


Figure 5 Concrete User interface view of the type user data task

A 3DUI is used for the final rendering for no particular reasons. The screenshot B (Figure 6) reproduces the worlds generated in Java3D where each container (*type user data*) is mapped onto the virtual space. All objects are then mapped recursively onto Java3D widgets depending on their data type. In this particular case, the designer selected also the graphical representation if any, along with the textual representing. In this visualization, we propose another way to represent the category selection the use of 3D personages instead of icons. This 3D graphic representation of the option could reinforce the understanding, notice that we keep the text below the personages. The screenshot A (Figure 6) illustrates the same UI but rendered in VRML. The difference is the absence of icons on the radio buttons. Notice in both cases the traditional view of a radio button was not used but the principle was kept.



Figure 6 The type user data task rendered in VRML and Java3D

## VI. CONCLUSION

In this paper a list of canonical UI task action types associated to task models is presented. The list is dual: a

verb describes the type of activity at hand; an expression designates the type of object on which the action is operated. By combining these two dimensions a derivation of interaction objects supposed to support a task becomes possible. In addition, task types provide some hints of the implementation of the system and in some cases the dialog of the application could be automatically generated when some patterns are identified. The remaining work must be towards a profound evaluation of this technique, whether is useful or not, or if provides advantages confronted to existing methods. Graphical UI (GUI) generation from task models is a recurrent solution in the literature [8][15][19][20]. Such solutions, target their solution to multi-device and multiplatform implementations. Then, this emphasizes the potential of relying on a structured approach in order to reach a UI that could be trace along with the models that specify it. Our future direction is on evaluating the possible volumes that can be used as widgets if the concretization of the User Interface is not restricted to two dimensions but extended to the third dimension. Finally, in this paper we have discussed properties of task without considering relationships among tasks, this has been previously reported in [10].

**Acknowledgments** We gratefully acknowledge the support of the 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), the Alban program supported by European Commission and the CONACYT program supported by the Mexican government.

#### REFERENCES

- [1] Bleser, T.W. & Sibert, John. "Toto: a tool for selecting interaction techniques". In: Proceedings of user interface software and technology (Snowbird, Utah, Oct.3-5,1990). New York: ACM, 1990, pp. 135-142.
- [2] Bodart, F., Vanderdonckt, J., On the Problem of Selecting Interaction Objects, Proc. of BCS Conf. HCI'94 "People and Computers IX" (Glasgow, 23-26 August 1994), G. Cockton, S.W. Draper, G.R.S. Weir (eds.), Cambridge University Press, Cambridge, 1994, pp. 163-178.
- [3] Calhoun, G. C.; Arbak, C. L. & Boff, K. R. "Eye-controlled switching for crew station design". In: *Proceedings of the Human Factors Society 28th annual meeting*, Santa Monica (CA): Human Factors Society, 1984, pp. 258-262.
- [4] Constantine L. L., "Canonical Abstract Prototypes for Abstract Visual and Interaction". In Proceedings of the 10<sup>th</sup> International workshop on Design, Specification and Evaluation of Interactive Systems DSV-IS 2003 (june 11-13 2003, Funchal, Portugal), LNCS 2844, Springer Verlag, Berlin, 2003, pp. 1-15.
- [5] Foley, V. Wallace and V. Chan, "The human factors of computer graphics interaction techniques", In IEEE Computer Graphics & Applications, (4), pp. 13-48 (1984).
- [6] Frank, M. and Foley, J. Model-based user interface design by example and by answering questions. In Proc. INTERCHI, ACM Conference on Human Factors in Computing Systems, (1993), 161-162.
- [7] Gaffar, A., Sinnig, D., Seffah, A., and Forbrig, P. 2004. Modeling patterns for task models. In Proceedings of the 3rd Annual Conference on Task Models and Diagrams (Prague, Czech Republic, November 15 - 16, 2004). TAMODIA '04, vol. 86. ACM, New York, NY, 99-104.
- [8] Gonzalez Calleros, J.M., Vanderdonckt, J. and J. Muñoz Arteaga, "A Method for Developing 3D User Interfaces of Information Systems", in Proc. of 6th Int. Conf. on Computer-Aided Design of User Interfaces CA-DUI'2006, Springer-Verlag, Bucharest, Rumania, 6-8 June 2006, pp. 85-100.
- [9] Greenstein, Joel S. & Arnaut, Lynn Y. "Input devices". In: M. Helander, (Ed.), Handbook of Human-Computer Interaction, Amsterdam: North-Holland, 1988, pp. 495-519.
- [10] Guerrero, J., Vanderdonckt, J., Gonzalez, J.M., *FlowiXML: a Step towards Designing Workflow Management Systems*, Journal of Web Engineering, 2008, to appear.
- [11] Hutchinson, Thomas E.; White, Jr., K.Preston; Martin, Worthy N.; Reichert, Kelly N. & Frey, Lisa A. "Human-Computer Interaction Using Eye-Gaze Input". IEEE Transactions on systems, man, and cybernetics, 19(6), 1989, p. 1527-1533.
- [12] Jansen, B. J. "Using Temporal Patterns of Interaction to Design Effective Automated Searching engines". Communications of the ACM. Vol. 49, No.4. pp. 72-74. April 2006.
- [13] Johnsgard, T.J., Page, S., R., Wilson, R.D., Zeno, R., J., A Comparison of Graphical User Interface Widgets for Various Tasks, Proceedings of the Human Factors & Ergonomics Society - 39th Annual Meeting, Human Factors and Ergonomics Society, octobre 1995., pp. 287-291.
- [14] Lenorovitz, D.R.; Phillips, M.D.; Ardrey, R.S. & Kloster, G.V. "A taxonomic approach to characterizing human-computer interaction". In: G. Salvendy (Ed.), Human-Computer Interaction. Amsterdam: Elsevier Science Publishers, 1984, pp.111-116.
- [15] Limbourg, Q., Vanderdonckt, J., Michotte, B., Bouillon, L., Lopez, V.: UsiXML: a Language Supporting Multi-Path Development of User Interfaces. In: Bastide, R., Palanque, P., Roth, J. (eds.) Engineering Human Computer Interaction and Interactive Systems. LNCS, vol. 3425, pp. 200-220. Springer, Heidelberg (2005).
- [16] Paternò, F., Santoro, C., One Model, Many Interfaces, in Proc. of CADUI'2002, pp. 143--154.
- [17] Paternò, F. Model-based design and evaluation of interactive applications. Applied Computing, Springer. 1999
- [18] Puerta, A.R. A Model-Based Interface Development Environment. IEEE Software 14, 4 (1997), pp. 41-47.
- [19] Stanculescu, A., Limbourg, Q. Vanderdonckt, J., Michotte, B. and Montero, F.: A transformational approach for multimodal web user interfaces based on UsiXML. *ICMI* (2005), 259-266.
- [20] Vanderdonckt, J.: A MDA-Compliant Environment for Developing User Interfaces of Information Systems. In: Pastor, Ó., Falcão e Cunha, J. (eds.) CAiSE 2005. LNCS, vol. 3520, pp. 16-31. Springer, Heidelberg (2005).