An Ontology-Based Method for Universal Design of User Interfaces

Elizabeth Furtado, João José Vasco Furtado, Wilker Bezerra Silva, Daniel William Tavares Rodrigues, Leandro da Silva Taddeo, Quentin Limbourg, and Jean Vanderdonckt**

Universidade de Fortaleza
NATI - Célula EAD
Washington Soares, 1321 – Bairo Edson Queiroz
Fortaleza (Ceará), BR-60455770 Brazil
Elizabet@feq.unifor.br, vasco@feq.unifor.br,
Wilker@unifor.br, Danielw@unifor.br,
Taddeo@unifor.br

*Université catholique de Louvain
Institut d'Administration et de Gestion
Place des Doyens, 1
Louvain-la-Neuve, B-1348 Belgium
Limbourg@qant.ucl.ac.be
Vanderdonckt@qant.ucl.ac.be

Workshop on Multiple User Interfaces over the Internet: Engineering and Applications Trends submission category: paper

Abstract: Universal design of user interfaces attempts to cover design issues in multiple contexts of use where multiple types of users may carry out multiple tasks, possibly on multiple domains of interest. Existing design methods do not necessarily support designing such user interfaces. A new design method is presented for this purpose in three layers: (i) a conceptual layer where a domain expert defines an ontology of concepts, relationships, and attributes of the domain of discourse; (ii) a logical layer where a designer specifies multiple models based on the previously defined ontology; and (iii) a physical layer where a developer derives multiple user interfaces from the previously specified models with alternatives.

Keywords: automated generation of user interfaces, computer-aided design of user interfaces, conceptual level, context of use, instantiation, logical level, model-based approach, modeling, ontology, physical level, universal design.

1. Introduction

Universal design [10] adheres to a vision where user interfaces (UIs) of interactive applications are developed for the widest population of users in different contexts of use by taking into account differences such as preferences, cognitive style, language, culture, habits, conventions, and system experience. Universal design of **single** or **multiple** UIs (MUIs) poses some difficulties due to the consideration of these multiple parameters depending on the supported differences. In particular, the multiplicity of parameters dramatically increases the complexity of the design phase by adding many design options among which to decide. In addition, methods for developing UIs do not mesh well with this variety of parameters as they are not necessarily identified and manipulated in a structured way nor truly considered in the design process.

The goal of this paper is to present a structured method addressing parameters required for universal design. The method **is** supported by a suite of tools all based on an ontology of the domain of discourse and models that capture instantiations of concepts identified in this ontology for producing **multiple UIs** for one design situation. These different UIs exhibit different presentation styles, dialogue genres, and UI structures.

The method structures the UI design in three levels of abstraction as represented in fig. 1:

1. The *conceptual level* enables a domain expert to define ontology of concepts, relationships, and attributes involved in the production of multiple UIs.
2. The *logical level* allows designers to capture requirements of a specific UI design case by instantiating concepts, relationships, and attributes with a graphical editor. Each set of instantiations results in a set of models for each considered design case (*n* designs in fig. 1).
3. The *physical level* helps developers in deriving multiple UIs from each set of models thanks to a model-based UI generator: in fig. 1, *m* possible UIs are obtained for UI design #1, *p* for UI design #2,…, *r* for UI design #*n*. The generation is then exported to imported in a traditional development environment for any manual edition (here, MS Visual Basic).

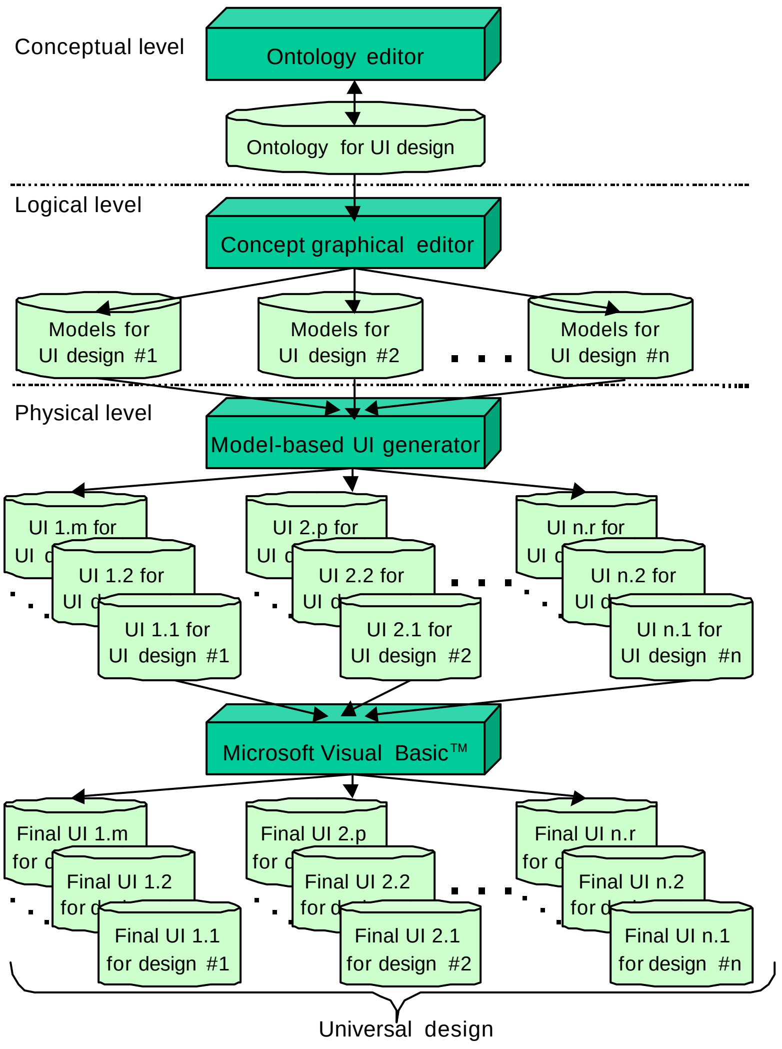


Figure 1. The different levels of the proposed method for universal design of user interfaces.

The remainder of this paper is structured as follows: section 2 provides a state of the art of methods for developing UIs with a focus on universal design. The three levels are progressively described: conceptual in section 3, logical in section 4, and physical in section 5. Although the method is generic, one particular application of this method is demonstrated throughout the paper by the usage of a particular series of supporting tools on the same case study: patient admission at a hospital. Each level is described by introducing motivation and goals, by presenting a supporting tool with required input/output, by showing how it addresses the case study, and by discussing some advantages of the layer and its associated tool. Section 6 summarizes the main points of the paper.

2. Related Work

The Author's Interactive Dialogue Environment (AIDE) [4] is an integrated set of interactive tools enabling developers to implement UIs by directly manipulating and defining its objects, rather than by the traditional method of writing source code. AIDE provides developers with a more structured way to develop UI than with traditional, yet radically different, "rush-to-code" approaches where unclear steps possibly result in a poorly usable UI.

User-Centered Development Environment (UCDE) [1] is an object-oriented UI development method investigating how software development can function as an extension of business process improvements. Business-oriented components (BOCs) are software objects that model business rules, processes, and data from the end-user's perspective. They clearly map this information onto UI objects that are compatible by construction with the information. The advantage of UCDE is a smooth process starting from high-level abstractions to final UIs.

Another methodological framework for UI development is provided in [6,7] which enables to integrate usability issues into the software development process from the beginning. The focal point of this approach is a psychologically based formal task description, which serves as the central reference for assessing the usability of the user interface under development. This contribution emphasizes the need of a task model as starting point for ensuring UI usability, whereas UCDE emphasizes the need of a domain model.

The MUSE method [8] uses structured notations to specify other elements of the context of use such as organizational hierarchies, conceptual tasks, and domain semantics. Moreover, graphical structured notations are proved to communicate UI design to users more easily.

In the above contributions, we see the importance of having a structured way to capture, store, and manipulate multiple elements of the context of use, such as task, domain, and user. Although the above methods partially consider this information, they do not consider designing multiple UIs where task [2,3], domain, and user parameters are varying, possibly simultaneously. Only the unified process [10] suggests deriving multiple refinements of a task model to cope with individual differences induced by universal design. However, this contribution is focussing more on task modeling operations than on steps and information required to progressively take multiple users in multiple contexts of use into account. The following method attempts to fill this gap by dividing the main problem into the three subsequent levels.

3. Conceptual Level

Motivation and goals. Each method to design UI displays its own set of concepts, relationships, and attributes, along with their possible values and ways to incorporate them in the method. However, this set is completely embedded in the method and its supporting tool, thus making the method little flexible to consider multiple parameters for universal design. When this set is hidden, the risk of manipulating fuzzy and unstructured pieces of information may occur. The conceptual level is therefore intended to enable domain experts to identify common concepts, relationships, and attributes involved in any particular way in universal design.

Description. An ontology can explicitly define any set of concepts, relationships, and attributes that need to be manipulated in a particular universal design [5,11]. The ontology notion comes from the Artificial Intelligence context where it is identified as the set of formal terms with one represents knowledge, since the representation completely determines what "exists" for the system. We hereby define a *context of use* as the global environment in which a user population, perhaps with different profiles, skills, and preferences, are carrying out a series of interactive tasks on one or multiple semantic domains. In universal design, it is expected to benefit from the advantage of considering any type of the above information to produce multiple UIs depending on the varying conditions. These pieces of information of a context of use can be captured in different models [9]. A *model* is hereby defined as a set of postulates, data and inferences presented as a declarative description of a UI facet. Many facets do exist as well as related models: task, domain, user, interaction device, computing platform, application, presentation, dialogue, help, guidance, tutorial, organizational environment. A model is typically built as a hierarchical decomposition of abstract concepts into several refined sub-levels. A model should also encompass relations between these concepts with roles, as well as for models, and between models.

Case study. For the simplicity of this paper, the context of use is focusing on three models:

1. A *domain model* defines the data objects that a user can view, access, and manipulate through a UI. These data objects belong to the domain of discourse. A domain model can be represented as a decomposition of information items, any item may be iteratively refined into sub-items. Each such item can be described by one or many parameters (such as data type, length). Each parameter possesses its own domain of possible values.
2. A *task model* is a hierarchical decomposition of a task into sub-tasks to end-up with actions which are no longer decomposed [8,11]. The model can then be augmented with temporal relationships stating when, how and why these sub-tasks and actions are carried out. Similarly to the domain model, a task model may hold a series of parameters with domains of possible values. For instance, task importance (low/medium/high), task structure (low/medium/high decomposition), task critical aspects (little/some/many), and required experience (low/moderate, high) are often considered.
3. A *user model* consists of a hierarchical decomposition of the user population into stereotypes. Each stereotype gathers people sharing the same value for a given set of parameters. Each stereotype can be further decomposed into sub-stereotypes. For instance, the population diversity may be reflected by many user parameters such as language, culture, preference (manual input vs. selection), task experience (elementary/medium/ complex), system experience (elementary/medium/complex), motivation (low/medium/ high), and experience of a complex interaction media (elementary/medium/complex).

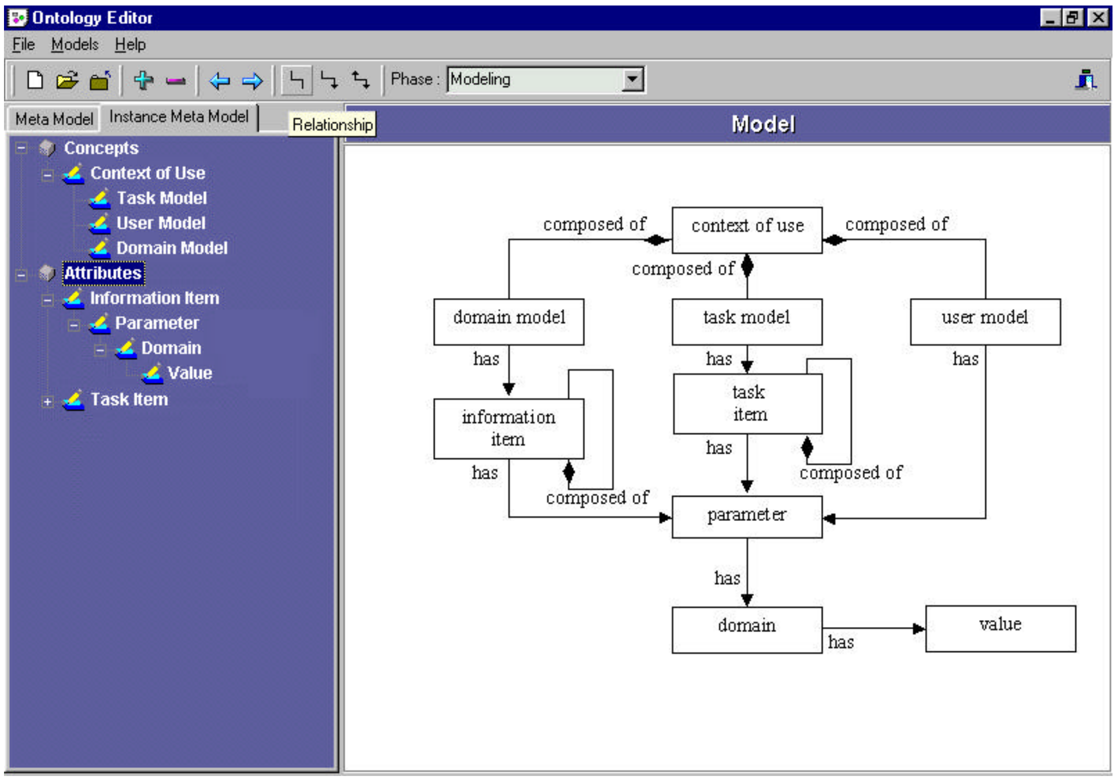


Figure 2. The ontology editor at the modeling stage.

Fig. 2 graphically depicts how the ontology editor can be used at the modeling stage to input, define, and structure concepts, relationships, and attributes of models used to describe a context of use. Here, the three models are represented and they all share a description by information parameters. Each parameter has a domain, each domain has a set of values, possibly enumerated. The composed-of relationship denotes aggregation, while has denotes properties.

Discussion. A definition of an ontology fosters structured UI design based on explicit concepts, relationships, and attributes rather than eclectic or extreme programming where the code is directly produced or rather than design methods which are not open to incorporate external information, such as required by universal design. Such ontology facilitates multidisciplinarity when people having different backgrounds need to gather for collaborative or participatory design. The **big win** of this level is that the ontology can be defined once and used as many times as wished. When universal design requires the consideration of more information in models or more models, this ontology can be updated accordingly and so is updated the method that supports universal design of UIs.

4. Logical level

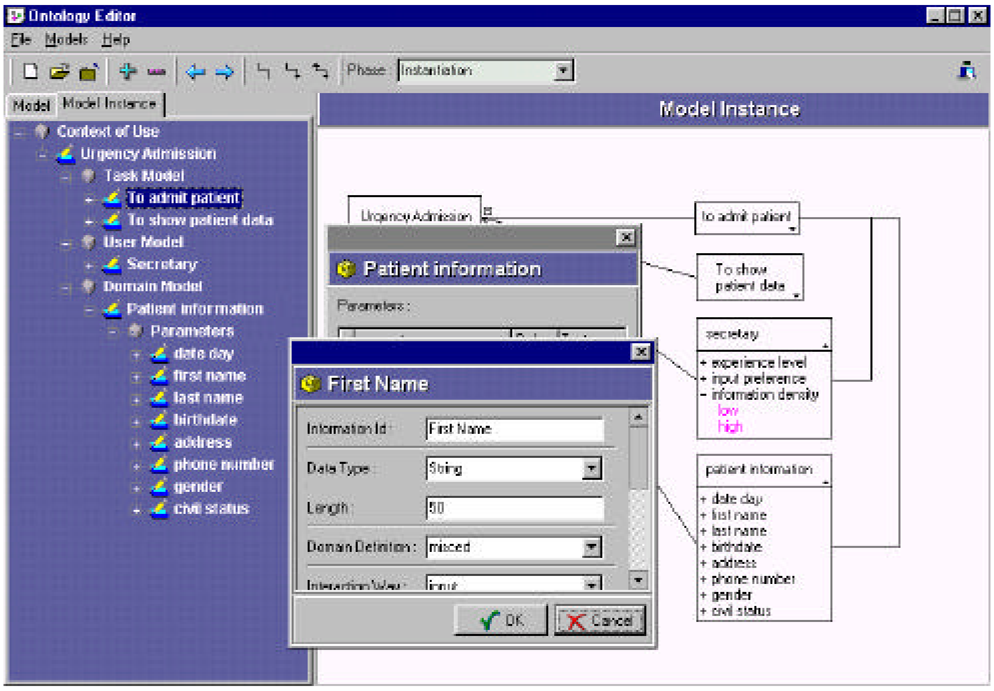
Motivation and goals. Each model defined at the conceptual level is now represented with its own information parameters. In the context of universal UI, a user model for instance is motivated by the observation that no unique UI may fit to an "average" user. Rather, multiple user stereotypes, stored as user models, allows the consideration of different user types in the same design. Any user modeling can be followed since there is no predefined/fixed set of parameters. This shows the generality of the method proposed here to support universal design.

Description. The set of concepts and attributes defined in the ontology are instantiated for each context of use of a domain. In this model-based approach, it means each model, which composes a context of use, is instantiated when its parameters are defined with domains of possible values.

Case study. The ontology editor is now used to instantiate the context of use, the relationships and attributes of models for the *Medical Attendance* domain involved in patient admission. Fig. 3 graphically depicts the *Urgency Admission* context of use and the attributes of models of task, user and domain. There are two tasks instantiated: *to admit patient* and *to show patient data*. The first one is activated by a *secretary* and uses *patient information* during its execution. To the user model of the secretary the parameters considered are: her/his experience level, input preference, information density with the enumerated values *low* and *high*.

The information items describing a patient are the following: date of the day, first name, last name, birth date, address, phone number, gender and civil status. Information items reagarding insurance affiliation and medical regimen can be described similarly. The parameters of an information item of a domain model depend on the UI design process.

For instance, parameters and values of an information item used to generate UIs in [12] are: data type (calendar, Boolean, graphic, integer, real, or alphanumeric), length (*n*>1), domain definition (know, unknown, or mixed), interaction way (input, output, or input/output), orientation (horizontal, vertical, circular, or undefined), number of possible values (*n*>1), number of values to choose (*n*>1), and precision (low or high).



Phase: Instantiation

Information Id: First Name
Data Type: String
Length: 50
Domain Definition: mixed
Interaction Way: input

Figure 3. The ontology editor at the instantiation stage.

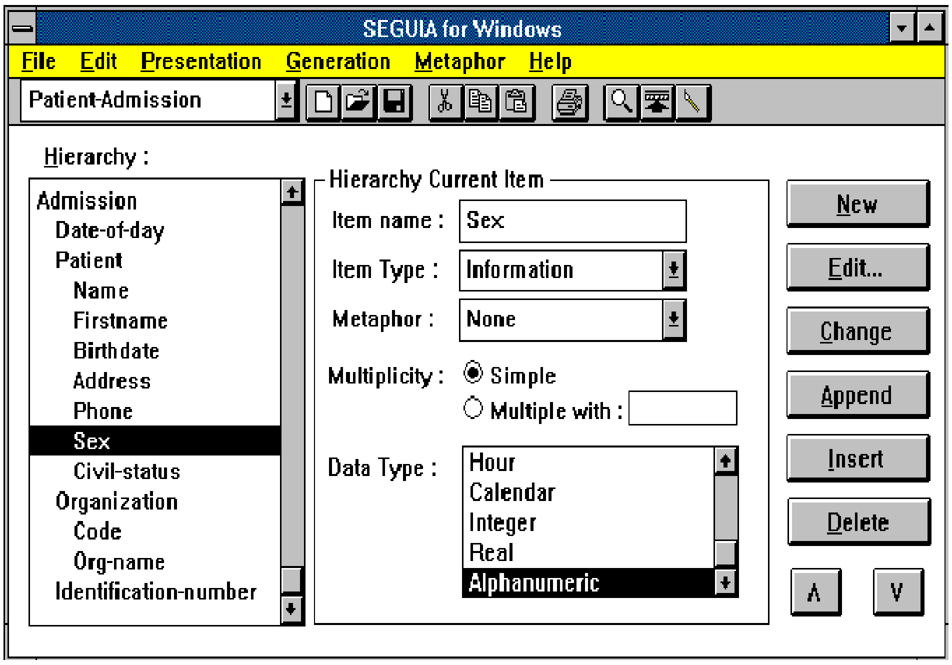
Discussion. Models defined and input at the logical level are all consistently based on the same ontology. The **big win** is that when the ontology changes, all associated models change accordingly since the ontology is used as a reference input for the graphical editor. The graphical nature of the editor improves the legibility and the communicability of information, while information which cannot be represented graphically is maintained in text properties. The models serve for both requirements documentation and UI production in the next level.

5. Physical level

Motivation and goals. The main goal of the physical level relies in its ability to exploit instantiations captured in individual models to produce multiple UIs, possibly for different computing platforms, development environments, or programming languages. This level is the only one which is dependent of the target hardware/software configuration intended to support the UI.

Description. Instantiations of the previously defined models, along with the values of their parameters, are stored in the logical level into specification files. Each specification file basically consists of a hierarchical decomposition of the UI models into models, parameters, values, etc. maintained in an ASCII file. This file can in turn be imported in different UI producers as needed. Here, we are using SEGUIA (fig. 4), a model-based interface development that is capable of automatically generating MS Visual Basic code for a running UI from any specification file. Of course, any other tool which is compliant with the model format and/or which can import the specification file may be intended to produce running UI for other design situations, contexts of use, user models, or computing platforms. SEGUIA is able to automatically generate several UI presentations to obtain multiple UIs. These different presentations are obtained

- In an automated manner, where the developer only launches the generation process by selecting which layout algorithm to rely on (e.g. two-column format or right/bottom strategy).
- In a computer-aided manner, where the developer can see at each step what are the results of the generation, can “intervene”, and govern the process before reaching a final status.



Patient-Admission

Hierarchy :

Admission
Date-of-day
Patient
Name
Firstname
Birthdate
Address
Phone
Sex
Civil-status
Organization
Code
Org-name
Identification-number

Hierarchy Current Item

Item name : Sex

Item Type : Information

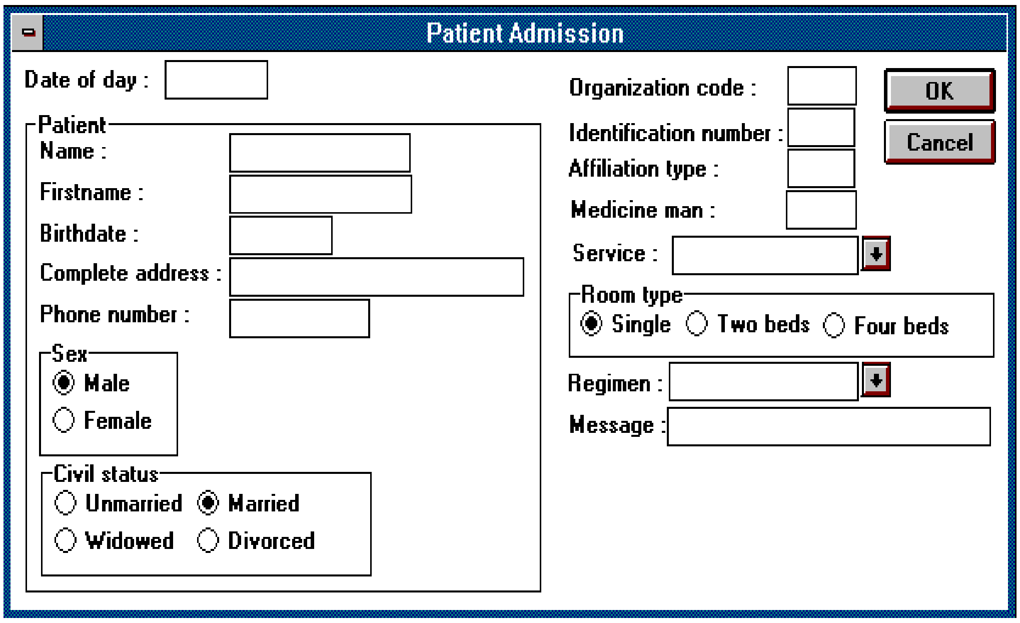
Metaphor : None

Multiplicity : [✓ Simple] Multiple with :

Data Type : Hour, Calendar, Integer, Real, [✓ Alphanumeric]

Figure 4. Example of a final UI for the considered case study during manual editing.

Case study. In our case study, information items and their values introduced at the instantiation level (fig. 3) are imported in a list box (left in fig. 4) of information items, each of which is specified separately (right in fig. 4). If needed, this information can still be edited (in fig. 4, for instance, *gender* has been renamed to *sex*). By selecting Generation in the menu bar, the developer launches the generation process and may produce different user interfaces, depending on rules used for this purpose. Selection rules automatically select concrete interaction objects (or *widgets*) from parameters and their values for each information item. For instance, this information item is mapped onto a radio-button with two values as illustrated in the first part of fig. 5. Selection rules are gathered in different selection strategies. Some strategies apply rules for selecting input/output widgets, while others prefer graphical representations. For example, in the second part of fig. 5, a graphical representation of the same information item has been preferred with reference to relevant icons. The selected widgets are then automatically laid out according layout algorithms based on presentation guidelines.



Date of day :

Patient
Name :
Firstname :
Birthdate :
Complete address :
Phone number :

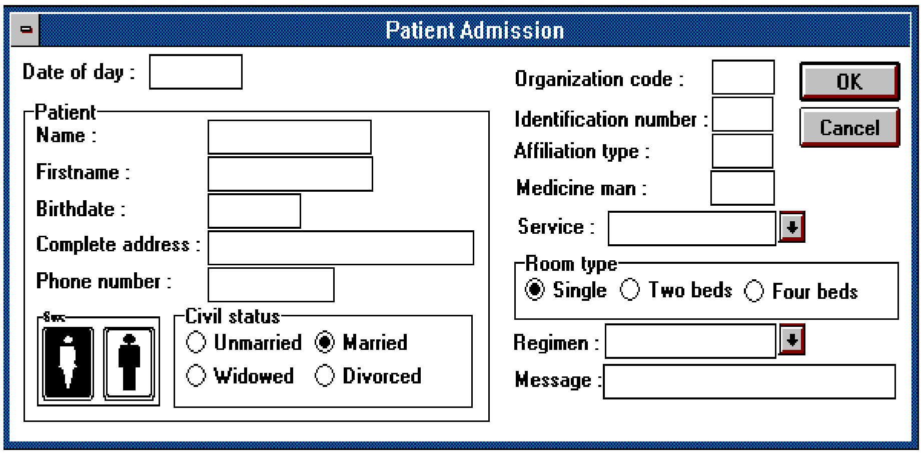
Sex
[✓ Male] Female

Civil status
Unmarried [✓ Married] Widowed Divorced

Organization code :
Identification number :
Affiliation type :
Medicine man :
Service :

Room type
[✓ Single] Two beds Four beds

Regimen :
Message :


Patient Admission

Date of day :

Patient
Name :
Firstname :
Birthdate :
Complete address :
Phone number :

Sex

Civil status
Unmarried [✓ Married]
Widowed Divorced

Organization code :
Identification number :
Affiliation type :
Medicine man :
Service :

Room type
[✓ Single] Two beds Four beds

Regimen :
Message :

OK
Cancel

Figure 5. Examples of different final UIs.

For instance, the first part of fig. 5 reproduces a rather straightforward presentation, whereas the second part optimised the presentation by moving the *Civil Status* group box and by resizing it so that it is properly aligned with other widgets.

Discussion. This level allows sharing or reusing previously defined models for several UI designs, which is particularly useful when working in the same domain where similar information can be found. It also encourages users to work at a higher level abstraction than merely the code level and to explore multiple UI alternatives for the same UI design case. This flexibility may even produce UIs with unforeseen, unexpected or under-explored features. The **big win** is that when the set of models change, all UIs that can be created from this set can change accordingly. The *design space* is often referred to as the set of all possible UIs that can be created from an initial set of models for one UI design.

6. Conclusion

The main contributions and benefits of the method presented in this paper are:

- UI design method can be explicitly structured into three separate levels (i.e., conceptual, logical, and physical, as frequently found in other disciplines such as data bases, software engineering, and telecommunications. Each level can be considered as a level of abstraction from the physical level as represented in fig. 6. The physical level is the instance level where instances of the case study are analysed, the logical level is the model level where theses instances are mapped onto relevant abstractions, and the conceptual level is the metamodel level where abstractions manipulated in the previous levels can be aggregated to identify the concepts, relationships, and attributes used in a particular method.

- The three levels make it possible to apply the "separation of concern" principle: (i) a definition of useful concepts first by someone who is aware of UI techniques such as user-centered design, task analysis, and human factors; (ii) a model definition where, for each UI design, multiple sets of models can be defined on the same basis wih no redefinition of previously defined concepts; and (iii) multiple UI creation: for each set of UI models, several UIs can be created by playing with parameters supported by the UI generator and manual editing is allowed when needed, thus achieving the goal stated in the introduction.

- The **big win** of the method is that change operated at any level are instantly propagated to subsequent levels: when the ontology changes, all possible models change accordingly; when a model change, all possible specification change accordingly and so the set of all possible UIs that can be created (the UI design space).

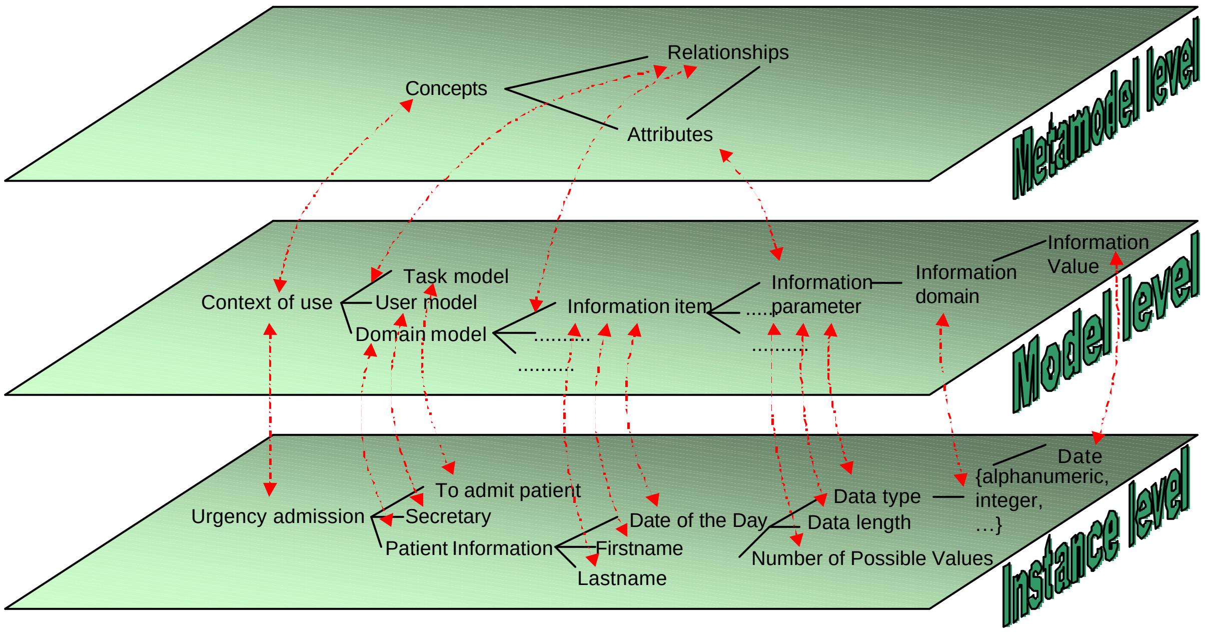


Figure 6. The different levels of the proposed method.

References

1. Butler, K.A., *Designing Deeper: Towards a User-Centered Development Environment Design in Context*, Proceedings of ACM Symposium on Designing Interactive Systems: Processes, Practices, Methods, & Techniques DIS´95 (Ann Arbor, 23-25 August 1995), ACM Press, New York, 1995, pp. 131-142.
2. Card, S., Moran, T.P., Newel, A., *The Psychology of Human-Computer Interaction*, Lawrence Erlbaum Associates, Pub., Hillsdale, 1983.
3. Gaines, B., *A Situated Classification Solution of a Resource Allocation Task Represented in a Visual Language*, Special Issue on Models of Problem Solving, International Journal of Human-Computer Studies, Vol. 40, No. 2, 1994, pp. 243-271.
4. Gimnich, R., Kunkel, K., Reichert, L., *A Usability Engineering Approach to the Development of Graphical User Interfaces*, Proceedings of the 4th International Conference on Human-Computer Interaction HCI International'91 (Stuttgart, 1-6 September 1991), Vol. 1, 1991, pp. 673-677.
5. Guarino, N., *Formal Ontology, Conceptual Analysis and Knowledge Representation: The Role of Formal Ontology in the Information Technology*, International Journal of Human-Computer Studies, Vol. 43, Nos. 5/6, 1995, pp. 625-640.
6. Hartson, H.R., Hix, D., *Human-Computer Interface Development: Concepts and Systems for its Management*, ACM Computing Surveys, Vol. 21, No. 1, 1989, pp. 241-247.
7. Hix, D., *Developing and Evaluating an Interactive System for Producing Human-Computer Interfaces*, Behaviour and Information Technology, Vol. 8, No 4, 1989, pp. 285-299.
8. Lim, K.Y., Long, J., *Structured Notations to Support Human Factors Specification of Interactive Systems Notations and Tools for Design*, Proceedings of the BCS Conference on People and Computers IX HCI'94, Cambridge University Press, Cambridge, 1994, pp. 313-326.
9. Puerta, A.R. *A Model-Based Interface Development Environment*, IEEE Software, Vol. 14, No. 4, July/August 1997, pp. 41-47. Accessible at http://www.arpuerta.com/pubs/ieee97.htm
10. Savidis, A., Akoumianakis, D., Stephanidis, C., *The Unified User Interface Design Method*, Chapter 21, in "User Interfaces for All: Concepts, Methods, and Tools" C. Stephanidis (ed.), Lawrence Erlbaum Associates, Pub., Mahwah, 2001, pp. 417-440.
11. Top, J., Akkermans, H., *Tasks and Ontologies in Engineering Modelling*, International Journal of Human-Computer Studies, Vol. 41, No. 4, 1994, pp. 585-617.
12. Vanderdonckt, J., Berquin, P., *Towards a Very Large Model-based Approach for User Interface Development*, Proceedings of 1st Int. Workshop on User Interfaces to Data Intensive Systems UIDIS'99 (Edinburg, 5-6 September 1999), N.W. Paton & T. Griffiths (eds.), IEEE Computer Society Press, Los Alamitos, 1999, pp. 76-85.