A Transformational Approach for Multimodal Web User Interfaces based on UsiXML

Adrian Stanciulescu, Quentin Limbourg, Jean Vanderdonckt, Benjamin Michotte and Francisco Montero
Université catholique de Louvain, School of Management (IAG)
Place des Doyens, 1 – B-1348 Louvain-la-Neuve, Belgium
+32 10/47{8349, 8525, 8384, 8379} - {stanciulescu, limbourg, vanderdonckt, michotte, montero}@isys.ucl.ac.be

ABSTRACT
A transformational approach for developing multimodal web user interfaces is presented that progressively moves from a task model and a domain model to a final user interface. This approach consists of three steps: deriving one or many abstract user interfaces from a task model and a domain model, deriving one or many concrete user interfaces from each abstract one, and producing the code of the corresponding final user interfaces. To ensure these steps, transformations are encoded as graph transformations performed on the involved models expressed in their graph equivalent. For each step, a graph grammar gathers relevant graph transformations for accomplishing the sub-steps. The final user interface is multimodal as it involves graphical (keyboard, mouse) and vocal interaction. The approach outlined in the paper is illustrated throughout a running example for a graphical interface, a vocal interface, and two multimodal interfaces with graphical and vocal predominances, respectively.

Categories and Subject Descriptors
D.2.2 [**Software Engineering**]: Design Tools and Techniques – *Computer-aided software engineering (CASE)*, *Evolutionary prototyping, Structured Programming, User Interfaces*. H.5.2 [**Information Interfaces and Presentation (e.g., HCI)**]: User interfaces – *Graphical user interfaces, Interaction styles, Input devices and strategies, Prototyping, Voice I/O*.

General Terms
Design, Human Factors, Standardization.

Keywords
Model-Driven Development, Multimodal interaction, Transformational approach, User Interface eXtensible Markup Language.

1. INTRODUCTION
While the W3C has made much progress in defining the W3C Multimodal Interaction framework [11] identifying the functional components of multimodal User Interfaces (UIs) and laying down the groundwork for the coordination and communication between these components at a browser implementation level, much more needs to be done in order to introduce a method for systematically developing multimodal UIs based on this framework [16] in a flexible way. Various methods have been proposed [14]:

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
ICMI'05, October 4–6, 2005, Trento, Italy.
Copyright ©2005 ACM 1-58113-000-0/00/0004…$5.00

1. **Type A:** multimodal and multi-device authoring (e.g., co-browser authoring) [4].
2. **Type B:** multimodal and multi-device authoring **(**e.g**.,** X+V [19] and SALT (http://www.saltforum.org)).
3. **Type C:** multimodal and multi-device authoring, where the interface is developed at the level of the data model (e.g., XForms in XHTML). The respective presentations for each modality or device are bound to the data model and manually authored or automatically generated from the data model [20].

While graphical [19] and vocal user interfaces for the Web [2] have been largely deployed according to those three types, *Multimodal Web User Interfaces* (MWUIs) remain less researched, investigated, and deployed [17]. Partly because they involve yet another new markup language that forces developers to (re)deploy applications according to this language. Also because the specifications and the design options involved in the development process turn to be unidentified, especially when it has to cope with the selection of what modality for which part of the interaction. Therefore, motivations for a Type C method for MWUIs include: the need for a systematic approach for developing such interfaces in forward engineering, to maintain high level models that have been used in non-multimodal Type C approaches constant and consistent for the new possibilities [13], to explore different design scenarios based on use cases [16], to support user-centered design by anchoring the approach in task and domain models, to deploy a wide array of MWUIs with different modalities, different properties [5] and design options, to accommodate the approach to various contexts of use. Based on the type C method, we consider MWUIs which are defined with:

- Four modalities are involved: M1 = (keyboard, command language), M2 = (mouse, direct manipulation), M3 = (microphone, restricted vocabulary-oriented natural language) and M4 = (loudspeakers, unrestricted natural language), where an interaction modality is defined as a couple (device, interaction language) [5]. The use of modalities could be either sequential when the modalities are used one after another or parallel when multiple modalities are used simultaneously (e.g., multiple devices used simultaneously, multiple interaction languages used simultaneously). A MWUI could be either sequential or parallel.
- No fusion/fission is needed: since an independent interpretation/rendering process for each modality is performed, there is no need to conduct a fusion of tokens in input to interpret the multimodal interaction and a fission in output.
- The multimodality type is exclusive: sequential and independent interaction channels are operated.
- Only Assignment and Equivalence are supported CARE properties [5]. Neither Complementarity nor Redundancy are re-

quired because no MWUI language can afford them.

The remainder of this paper is structured as follows: Section 2 summarizes related work that attempt to address the same motivations as explained above for MWUIs. Section 3 outlines the transformational approach that is developed here and their underling concepts structured in four layers. Section 4 details these three steps and illustrate them throughout a running example. Section 5 concludes the paper by reporting on the benefits of the approach with respect to existing state of art.

2. RELATED WORK

A multitude of multimodal interactive systems has already been developed as off-line applications (e.g., [15, 17]) or on-line applications for the Web (e.g., [11, 20, 21]). Several separate requirements have been identified in these works: usage of models to produce the multimodal interface (e.g., [2, 6, 18]), description of these models with a specification language (e.g., CTL [1], UIML [19], XISL [10]), explicit design options for multimodal dialog (e.g., for help, CARE properties [5]), task-based design of multimodal applications [4]). We are not aware of any work that combines all these requirements into one single systematic approach.

Vida Software's Natural Interaction platform (http:// www.vida-software.com) is dedicated to interfaces that consider different isolated modalities restricted to the markup languages, such as WML, HTML, XHTML, and VoiceXML, but does not integrate them into one single interface. MONA (Mobile multimOdal Next generation Applications) [17] 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 (e.g., mobile phones and PDAs). However, they do not have any systematic approach for developing such applications based on models, particularly the user's task and a specification language.

EMMA (Extensible Multimodal Annotation markup language - http://www.w3.org/TR/emma/) is intended for use by systems that provide semantic interpretations for a variety of inputs, including but not necessarily limited to, speech, natural language text, GUI and ink input. This markup will be used primarily as a standard data interchange format between the components of a multimodal system; in particular, it will normally be automatically generated by interpretation components to represent the semantics of users' inputs, not directly authored by developers. As such, EMMA does not represent a specification language such as UIML [19] or XISL [10], and does not contain any transformational approach that initiates a progressive development from different models.

Teresa [2] separately generates either a graphical UI or a vocal UI for multiple platforms.

Our transformational approach is different of existing approaches in that all the design knowledge that is required to conduct the transformations is explicitly given in transformation rules. The execution of these rules is ensured by a transformation engine that is separate from the transformation logic, as opposed to existing systems where it is implicit. The originality of our approach is given also by the way in which the designer may explore many design alternatives and produce several UIs exhibiting various modalities (up to 4 different) without restarting the process from the beginning.

3. OUTLINE OF THE TRANSFORMATIONAL APPROACH

3.1 Reference Framework for Multi-target UIs

The foundation of the transformational approach for MWUIs that is presented in this paper is that all the information pertaining to the models describing the future MWUI is specified in the same User Interface Description Language (UIDL) throughout the development life cycle. This UIDL is UsiXML (User Interface eXtensible Markup Language – http://www.usixml.org) and consists of a UIDL characterized by the following principles:

- *Expressiveness of UI*: any UI is expressed depending on the context of use thanks to a suite of models that are analyzable, editable, and manipulable by a software agent.
- *Central storage of models*: each model is stored in a model repository where all UI models are expressed similarly.
- *Transformational approach*: each model stored in the model repository may be subject to one or many transformations supporting various development steps. Each transformation is itself specified thanks to UsiXML [13].

Contrarily to UIML [19] and XISL [10], UsiXML [13] enables the specification of all models and the transformations between them until a final MWUI is obtained. UsiXML is able to specify various UIs with the four modalities of interaction defined in Section 1. For this purpose, UsiXML is structured according to four basic levels of abstractions defined by the Cameleon reference framework [3] (Fig. 1).

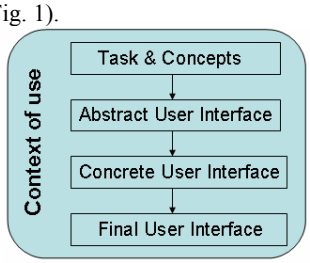

Figure 1. The Cameleon reference framework for multi-target UIs.

At the top level is the ***Task & Concepts*** level that describes the various interactive tasks to be carried out by the end user and the domain objects that are manipulated by these tasks. These objects are considered as instances of classes representing the concepts.

An ***Abstract UI*** (AUI) provides a UI definition that is independent of any modality of interaction (e.g., graphical interaction, vocal interaction, etc.). An Abstract UI is populated by *Abstract Containers* (AC), *Abstract Individual Components* (AIC) and abstract relationships. AICs represent basic system interactive functions, which are referred to as *facets* (input, output, navigation, control). In this sense, AICs are an abstraction of widgets found in graphical toolkits (like windows, buttons) and in vocal toolkits (like vocal input and output widgets in the vocal interface). Two AUI relationships that can be defined between AICs:

1. *Dialog transition*: specifies a navigation transition within a abstract container or across several abstract containers.
2. *Spatio-temporal relationship*: characterizes the physical constraints between AICs as they are presented in time and space.

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.

The ***Concrete UI*** (CUI) concretizes an AUI for a given context of use into *Concrete Interaction Objects* (CIOs) so as to define layout and/or interface navigation of 2D graphical widgets and/or vocal widgets. Any CUI is composed of CIOs, which realize an abstraction of widgets sets found in popular graphical and vocal toolkits (e.g., Java AWT/Swing, HTML 4.0, Flash DRK6, VoiceXML, and VoxML). A CIO is defined as an entity that users can perceive and/or manipulate (e.g., push button, text field, check box, vocal output, vocal input, vocal menu). The CUI abstracts a Final UI in a definition that is independent of programming toolkit peculiarities.

Because UsiXML considers both graphical and vocal modalities, CIOs are further divided into two types: graphicalCIOs and vocalCIOs. Each of this type is further divided into Containers and Individual Components. *Graphical containers* (GC) (e.g., window, table, dialog box) contain a collection of *Graphical Individual Components* (GIC) (e.g., button, text component, menu), while *Vocal Containers* (VC) (e.g., vocalForm, vocalMenu, vocalConfirmation) are composed of a collection of *Vocal Individual Components* (VIC) (e.g., vocalFeedback, vocalPrompt, vocalMenuItem, vocalInput).

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).

The ***Context of use*** 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 contained in a separate model: *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. In order to map different elements belonging to the models described above, UsiXML provides the designer with a set of pre-defined relationships called *mappings*. These mappings are used throughout the steps of the transformational approach [12, 14]:

- *Manipulates*: maps a task onto a domain concept.
- *Updates*: maps an interaction object and a domain model concept (specifically, an attribute).
- *Triggers*: maps an interaction object and a domain model concept (specifically an operation).
- *Is Executed In*: maps a task onto an AUI or CUI element.
- *Is Reified By*: maps an abstract object into a concrete one through an abstraction transformation.

3.2 Specification of Transformations

To progressively move from the uppermost level, the "Task & Concept" level, to the bottom level, the "Final UI", a transformational approach suggests that each development step can be achieved through applying a series of transformations. A transformation applies transformation rules on initial models so as to produce the resulting models. To specify such transformations, UsiXML is equipped with an underlying graph structure thanks to which all models and transformations between them can be described to support model transformation. Therefore, a *transformation system* is composed by a series of *transformation rules*, which are in turn expressed in rules between graph structures (Fig. 2).

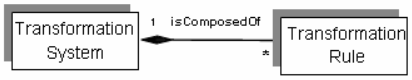

Figure 2. Development elements of the transformational approach.

Fig. 3 illustrates how a transformation system applies to any UsiXML specification: let G be the initial UsiXML specification, when 1) a ***Left Hand Side*** (LHS) matches into G and 2) a ***Negative Application Condition*** (NAC) does not match into G (several NAC may be associated to a rule), 3) the LHS is replaced by a ***Right Hand Side*** (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. Variables may be associated to attributes within a LHS. An expression may compare this variable with a constant or with another variable.

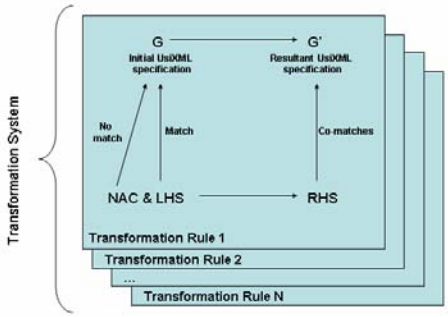

Figure 3. Characterization of transformation in UsiXML.

The transformation approach is sustained by **TransformiXML Enviroment** that allows the definition and the application of transformation rules. This environment is sub-divided into two components: a Java Application Programming Interface (TransformiXML API) that can be used by any application to apply transformation rules and a Graphical User Interface that serves as a front-end application to the API (TransformiXML GUI). Attributed Graph Grammars (AGG) API (http://tfs.cs.tu-berlin.de/agg/) performs model-to-model transformations. The basic flow of tasks with TransformiXML GUI (Fig. 4) 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). Here the starting point is the task and domain model 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 systems contain a set of rules that can be visualized in the **Transformation rule explorer** frame. The user may also want to edit the rules either in an XML editor (the one of GrafiXML, for instance) or in AGG environment. The user may apply the transformation either step by step or as a whole. The result of the transformation is then explicitly saved in a UsiXML file.

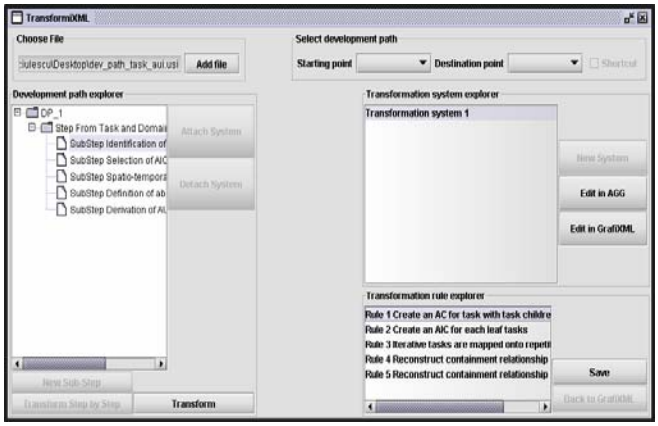

Figure 4. TransformiXML GUI.

The four levels of the reference framework and the mechanism supporting the forward engineering from the "Task & Concepts" level to the "Final UI" level allows defining the four steps of the transformational approach:

1. The task and domain models are specified first so as to initiate the forward engineering.
2. In order to produce one or many AUIs that are independent of any modality, TransformiXML applies model-to-model transformations (here, task & domain to AUI) to realize this step.
3. From each AUI, different CUIs can be obtained similarly thanks to model-to-model transformations (here AUI to CUI). Each concrete UIs can specify 2D or 3D graphical UI or multimodal UI. Each can be targeted to a particular platform.
4. From each CUI, a corresponding *Final UI* (FUI) can be produced by automated generation of code from the models. In order to fulfill this task, the GrafiXML editor is used for graphical UIs, and XSL transformations are used for the vocal and multimodal UIs. One or many MWUIs are then obtained. XHTML code can be generated for graphical UIs, VoiceXML code can be generated for vocal interfaces and XHTML+Voice (X+V) for multimodal UIs. X+V represents a unified standard for multimodal interfaces so that applications can be written once and used in different environments, including Web pages, telephones and handheld devices.

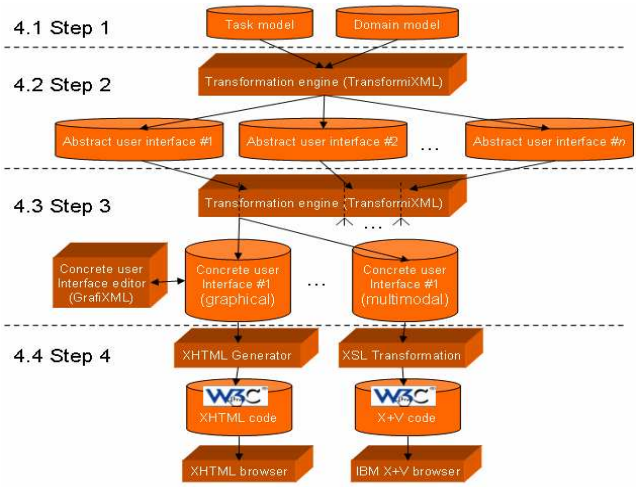

Figure 5. General development scenario of UI.

4. THE FOUR STEPS OF THE TRANSFORMATIONAL APPROACH

To exemplify the transformational approach, a running example is selected basically for understanding purposes: an opinion polling system aiming at collecting opinions of users regarding a certain subject. The scenario of this example is based on the general development scenario described above: from the domain and task model, an AUI is produced from which many CUIs are derived (2D graphical UI, vocal UI and multimodal UI). In the last step, several FUIs are derived from the CUI.

4.1 Step 1: The Task and Domain Models

The task model, the domain model and the mappings between them are graphically described using IdealXML [16], an Interface Development Environment for AppLications specified in UsiXML.

The upper part of Fig. 6 depicts a *CTT* (Concurred Task Tree) representation [2] 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 Fig. 6 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. IdealXML generates automatically the UsiXML specifications for the task and domain model edited graphically with the help of the tool.

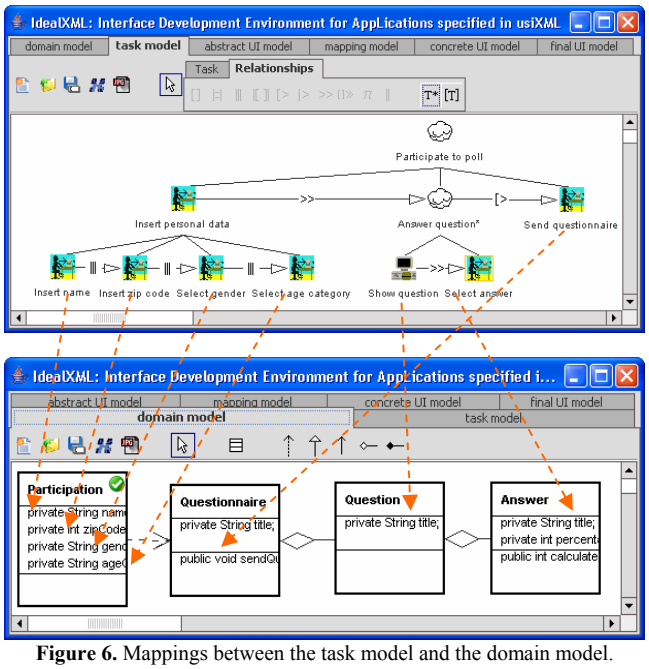

Figure 6. Mappings between the task model and the domain model.

The dashed arrows between the two models in Fig. 6 depict the model mappings, such as **manipulates** relationships between 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.** Fig. 7 illustrates the mapping model between the task model and the domain model. Each of the tasks is mapped on the corresponding attribute or method of the classes contained in the domain model. IdealXML automatically generates the UsiXML specifications of the mapping model.

4.2 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.

Rule 1. Create an AC for task with task children (Fig 8). The LHS contains two nodes representing two tasks, task (2) being the decomposition of task (1). The NAC specifies that the decomposed task (1) is executed into an AC, while the RHS recreates the structure of LHS adding an AC in which the decomposed task will be executed.

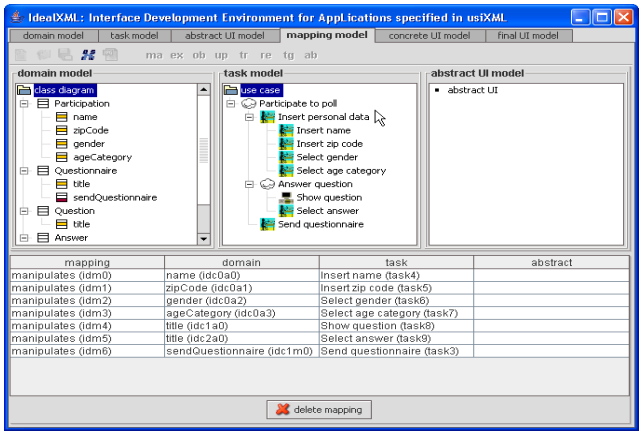

Figure 7. Mappings for the opinion polling system.

The application of this rule on the task model represented in the form of a graph G is the following: when the LHS matches into G and the NAC does not match into G, the LHS is replaced by the RHS, resulting G'.

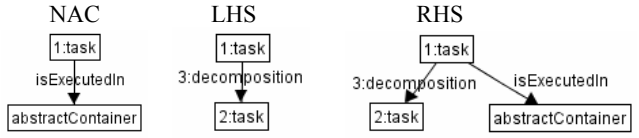

Figure 8. Create an AC for task with task children.

Rule 2. Create an AIC for each leaf task (Fig 9). The LHS contains a node representing the leaf task (1). The NAC specifies that the task (1) is executed into an AC, while the RHS creates an AIC in which the task (1) will be executed. Following the same mechanism of rule application described for Rule 1, Fig. 9 describes a rule which creates an abstract individual component (AIC) for each leaf task found in the task model.

Each AIC can be equipped with facets describing its main purpose/functionality. These facets are derived from the combination of task model, domain model and the mappings between them.

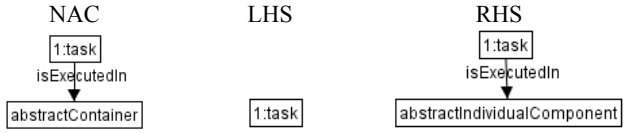

Figure 9. Create an AIC for each leaf task.

Rule 3. Create input facet for AICs executed in creation tasks (Fig.10). AICs that executes task for which the value of the attribute **userAction** is **create** and the value of **task item** attribute is **element,** are equipped with an input facets of type **create attribute value** (**create name**, **create zipCode**).

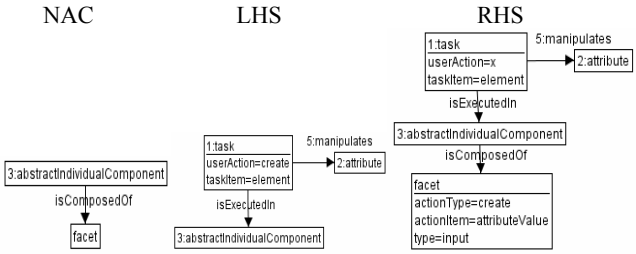

Figure 10. Create an input facet for AICs executed in creation tasks.

Depending on the values of attributes **user action** and **task item**, each specific type of task executed into an AIC determines the design of a corresponding rule in order to provide AIC with facet of type **select** (**select Gender**, **select ageCategory**, **select Answer**), **output** (**output Question**) or **control** (**send Questionnaire**). The main objective of UsiXML is to provide a machine processable language and then a human readable specification. Thus, the AUI of the virtual polling system is obtained by executing in TransformiXML a set of transformation rules and can be graphically depicted within the IdealXML (Fig.11). Rule 1 and Rule 2 are illustrated in Fig. 4. Fig. 12 reproduces the UsiXML specification for the AC **Answer question** which contains two AIC **(Output Question** and **Select Answer),** each one having its own corresponding facet (**Output Question** facet and **Select Answer** facet)**.**

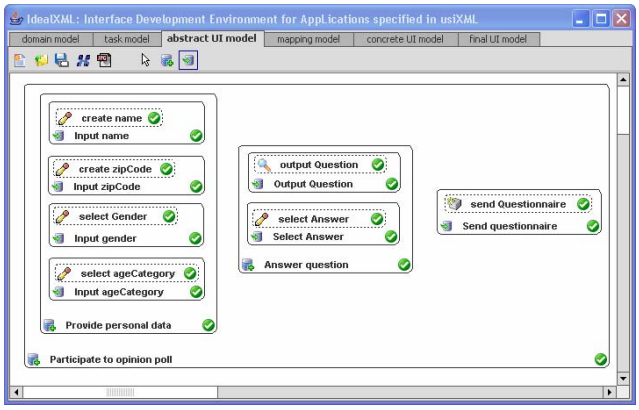

Figure 11. AUI of virtual polling system.

```
<abstractContainer id="idao2" name="Answer question">
    <abstractIndividualComponent id="idao11" name="Output Question">
        <output id="idao17" name="Output Question" actionType="abstract" actio-
        nItem="attribute value" />
    </abstractIndividualComponent>
    <abstractIndividualComponent id="idao12" name="Select Answer">
        <input id="idao18" name="Select Answer" actionType="select" actio-
        nitem="attribute"/>
    </abstractIndividualComponent>
</abstractContainer>
```

Figure 12. UsiXML specification for an AC

Rule 4. Abstract dialog derivation from task model (Fig 13). For each couple of sister tasks mapped onto AICs, a dialog control relationship will be established. The dialog control relationship has the same semantic as the temporal relationship. Following the same logic, for each combination of AC and AIC, a specific rule is defined.

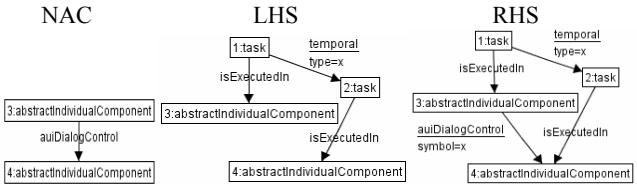

Figure 13. Abstract dialog derivation from task model.

4.3 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. Total graphical UI i.e., the modality used to interact with the system is entirely graphical (monomodal UI).
2. Predominant graphical UI i.e., the user fulfills her task with more graphical interaction than the vocal one (multimodal UI).
3. Predominant vocal UI i.e., the vocal modality is present in a higher proportion then the graphical one (multimodal UI).
4. Total vocal UI i.e., the modality used to interact with the system is entirely vocal (monomodal UI).

For each type of CUI a transformation system containing specific rules is provided. In the following it will be emphasized the modularity and the extensibility of transformation rules applied in order to obtain the desired CUIs. The selection of concrete interaction components involves a high number of rules due to the numerous different combinations of facet types, data types, cardinalities, etc.

Rule 5. Create radioButtons and vocalMenuItems for each selection value of a facet (Fig. 14, 15 and 16**).** The graphical part of the rule (depicted in red) illustrates respectively the NAC, LHS and RHS of the rule applied in order to obtain a group of radio buttons for the total graphical UI. These radio buttons will allow users to select the gender, age category and their answers to the questions. The NAC specifies that a selection value (5) is reified into a GIC. The graphical part of LHS (Figure 15) describes an AC (1), reified by a GC (2) and containing an AIC (3) named **y**. The AIC is composed of a facet (4) of type **select**, which, at his turn, is composed of a selection value (5) stored in variable **x**. Moreover, the AIC is reified by a GC (6) of type **horizontal box** that is contained into GC (2). The graphical part of RHS (Figure 16) recreates the structure of LHS and adds a GIC of type **radio button** that reifies the associated selection value (5). The GIC is contained by GC (6), has the default content of the associated selection value and is added in a group of radio buttons named after the AIC (3). Considering the initial UsiXML representation in the form of a graph G, the application of the above described rule is the following: when the LHS matches into G and the NAC does not match into G, the LHS is replaced by the RHS, resulting a transformed graph G'. If the designer wants to obtain a multimodal interaction with the system, the rule described above can be easily extended with new components. In the following, we will show the modularity and the extensibility of transformation rules by describing how vocal components (depicted here in blue) are added to the already existing graphical components, thus creating multimodal UIs. A new rule is obtained and used to provide a group of graphical radio buttons and the associated vocal menu items allowing users to have a graphical output feedback as a result of a vocal input.

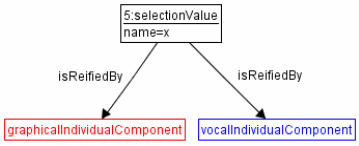

Figure 14. NAC of multimodal individual component rule.

To the already existent graphical NAC (Fig. 14) a VIC that reifies the selection value (5) is added. The LHS (Figure 15) is extended with a VC (7) of type **vocalMenu** that contains a VIC (8) of type **vocalInput**. In extension to the already existing structure described in Figure 16, VIC of type **vocalMenuItem** that is the reification of the selection value (5) is added. The defaultContent of the vocalMenuItem contains the reified selection value. The mappings between nodes and between edges belonging to the three components of the rule (NAC, LHS, RHS) are specified by attached numbers. The execution of this rule follows the same mechanism described for previous ones.

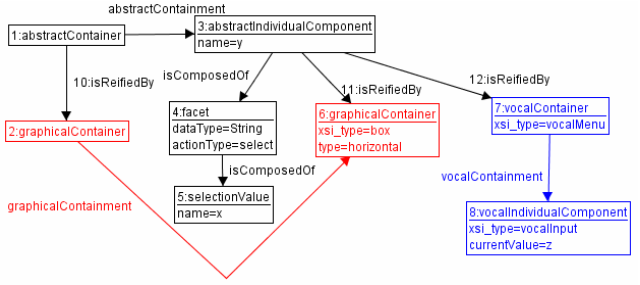

Figure 15. LHS of multimodal individual component rule.

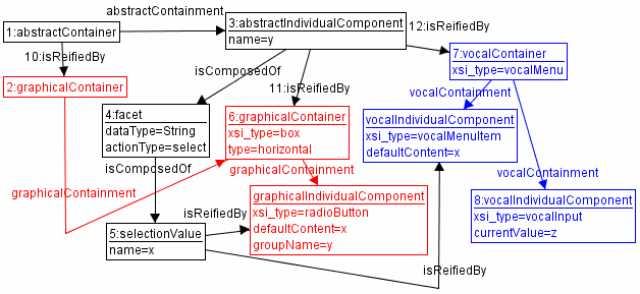

Figure 16. RHS of multimodal individual component rule.

In order to obtain a total vocal interaction a simple mechanism that eliminates the graphical components of the above rule can be employed. Only the vocal components and their relationship with the abstract components described in Figures 14, 15 and 16 should be kept.

Fig. 17 shows the UsiXML textual expression of the transformation rule described above. The graphical components are emphasized with a red color while the vocal components are in blue. The mappings between different components within the NAC, LHS and RHS parts of a rule are realized by using the **ruleSpecificId** attribute value of each component as a source or target of the mapping relationship. The **ruleMapping** element realizes the mappings between the LHS and RHS and between the LHS and NAC.

```
<transformationRule id="C_MM_RB" name="Creation of multimodal radio buttons">
    <nac>
        <selectionValue ruleSpecificId="N1"/>
        <graphicalIndividualComponent ruleSpecificId="N2"/>
        <vocalIndividualComponent ruleSpecificId="N3"/>
        <isReifiedBy ruleSpecificId="N4">
            <source sourceId="N1"/>
            <target targetId="N2"/>
        </isReifiedBy>
        .....
    </nac>
    <lhs>
        <abstractContainer ruleSpecificId="L1">
            <abstractIndividualComponent ruleSpecificId="L2" name="y">
                <facet ruleSpecificId="L3" dataType="String" actionType="select">
                    <selectionValue ruleSpecificId="L4" name="x"/>
                </facet>
            </abstractIndividualComponent>
        </abstractContainer>
        <graphicalContainer ruleSpecificId="L5">
            <graphicalContainer ruleSpecificId="L6" xsi_type="box" type="horizontal"/>
        </graphicalContainer>
        <vocalContainer ruleSpecificId="L7" xsi_type="vocalMenu">
            <vocalIndividualComponent ruleSpecificId="L8" xsi_type="vocalInput" current-
            Value="z"/>
        </vocalContainer>
        .....
    </lhs>
    <rhs>
      <abstractContainer ruleSpecificId="R1">
        <abstractIndividualComponent ruleSpecificId="R2" name="y">
            <facet ruleSpecificId="R3" dataType="String" actionType="select">
                <selectionValue ruleSpecificId="R4" name="x"/>
            </facet>
        </abstractIndividualComponent>
      </abstractContainer>
      <graphicalContainer ruleSpecificId="R5">
        <graphicalContainer ruleSpecificId="R6" xsi_type="box" type="horizontal">
            <graphicalIndividualComponent ruleSpecificId="R7" xsi_type="radioButton" de-
            faultContent="x" groupName="y"/>
        </graphicalContainer>
      </graphicalContainer>
      <vocalContainer ruleSpecificId="R8" xsi_type="vocalMenu">
        <vocalIndividualComponent ruleSpecificId="R9" xsi_type="vocalInput" current-
        Value="z"/>
        <vocalIndividualComponent ruleSpecificId="R10" xsi_type="vocalMenuItem" de-
        faultContent="x"/>
      </vocalContainer>
      .....
      </rhs>
      <ruleMapping sourceId="L1" targetId="R1"/>
      .....
</transformationRule>
```

Figure 17. Textual expression of transformation rules.

Rule 6. Create multimodal text component (Figs. 18 and 19). The rule is applied in order to create concrete interaction components of type text component that will allow users to input their **zip code** using the vocal modality, while the system's feedback to the recognized speech will be graphical. The created structure is used for the predominant vocal UI. The obtained graphical components (Fig.19) are represented in red (a non-editable text component representing a label, an editable text component representing a text field and the GC that contains them) while the vocal components are represented in blue (a vocal prompt, a vocal input and the VC of type **vocalForm** containing them). If the designer wants to obtain a rule that will allow a monomodal interaction (graphical or vocal), only the corresponding parts of the rule should be chosen, the abstract level and the associated mappings remaining unchanged.

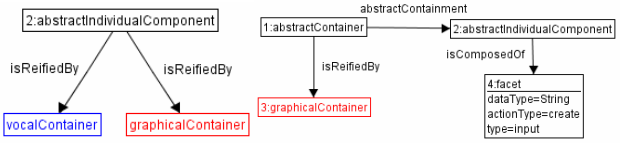

Figure 18. NAC, LHS of rule creating a multimodal text component.

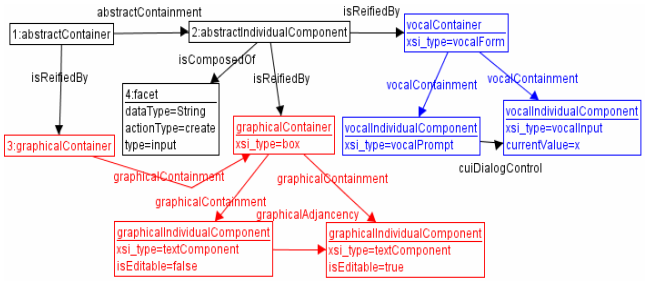

Figure 19. RHS of rule creating a multimodal text component.

The structure of the CUI appearance for the predominant vocal UI of the Virtual Polling System is illustrated in Fig. 20. The predominance of UI is given by the interaction modality used to fulfill the tasks. Thus, the proportion of vocal modality is higher then the graphical one. By vocal modality, we understand modality M3 described in the introduction of this paper, while M1 and M2 are considered graphical modalities. In **Provide Personal Data** section of Fig. 20, graphical modality is assigned only to the **create name** task. For the rest of the tasks (**create zipCode**, **create gender** and **select age**) only the vocal modality is assigned to be used in input. As a result of the speech recognition of the vocal input, a graphical feedback is provided in the associated GIC. In the design of the CUI, the use of vocal modality is emphasized by a microphone, offering a graphical guidance to the user. In the **Quesionnaire** section the vocal modality is represented by the use of modality M4 in order to provide a vocal output to the user (the system is uttering the question). The vocal guidance offered to the user is emphasized here by an icon symbolizing a loudspeaker. In order to select the answer the vocal modality is assigned too. Afterwards, a vocal confirmation is provided to the user. The **Send Questionnaire** task is fulfilled by using the graphical modality M2.

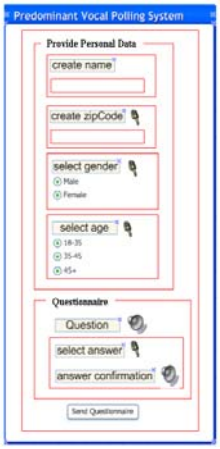

Figure 20. Predominant vocal UI.

4.4 Step 4: From CUI Model to FUI Model

The last step consists of transforming each variant of CUI into their respective FUI. The total graphical UI is obtained by using the XHTML generator of the GrafiXML editor. The resultant XHTML code is further interpreted by any XHTML browser, obtaining the FUI in the Fig. 21. The two multimodal CUIs (predominant graphic and predominant vocal UIs) and the total vocal UI are submitted to a XSL Transformation in order to obtain the X+V, respectively the VoiceXML code. The X+V code is further interpreted with NetFront X+V browser, one of the IBM WebSphere Multimodal Browser, while the VoiceXML code is interpreted with IBM VoiceXML browser. The resultant multimodal FUI of the predominant graphical and predominant vocal UI are shown in Figs. 22 and 23. Fig. 24 is a textual representation of total vocal UI (C = Computer and U = user). The total graphic UI can be obtained not only for the web but also for other targets, such as Visual Basic (Fig. 25).

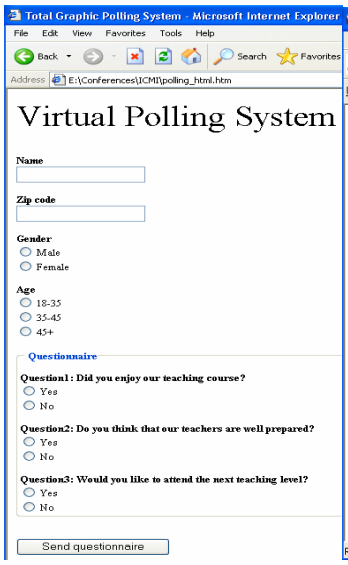

Figure 21. Total graphical FUI.

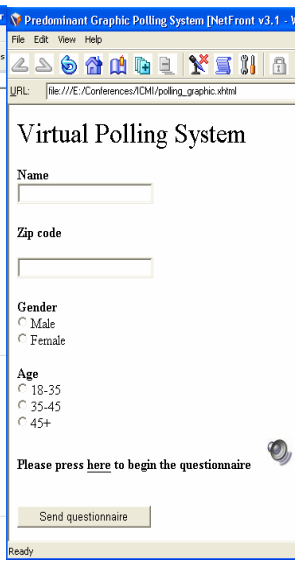

Figure 22. Predominant graphical FUI.

5. CONCLUSION

A transformational approach for developing MWUIs has been presented. The approach relies on a reference framework that decomposes the UI development life cycle into four levels. The transformational approach consequently structures the development into four steps, each step performing a transformation from the previous level to the next level until a final UI is reached.

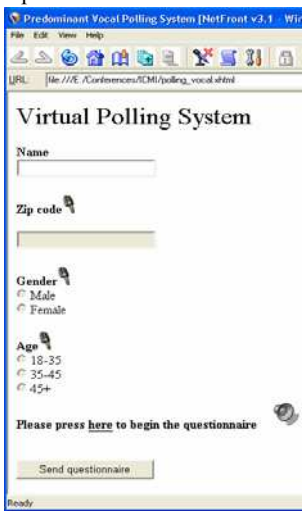

Figure 23. Predominant vocal FUI.

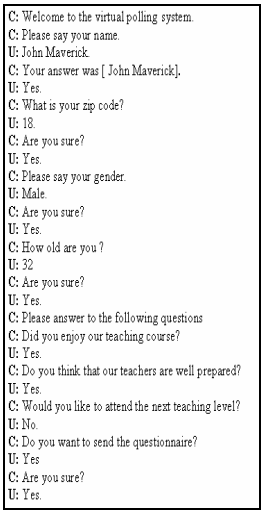

Figure 24. Total vocal FUI.

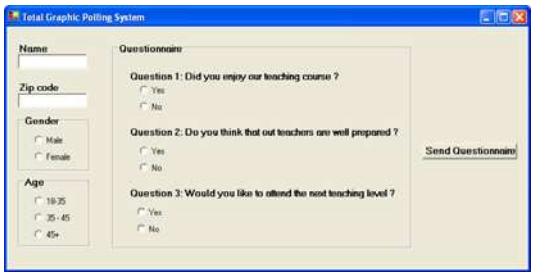

Figure 25. Visual Basic representation of total graphic FUI.

All elements, models, and transformations between these levels have been continuously and uniformly specified thanks to a single UIDL: UsiXML. Therefore, this transformational approach is superior to existing approaches in that all the design knowledge that is required to conduct the transformations is explicitly given in transformation rules. The execution of these rules is ensured by a transformation engine that is separate from the transformation logic, as opposed to existing systems where it is implicit. In this way, the designer may explore many design alternatives and produce several UIs exhibiting various modalities (up to 4 different) without restarting the process from the beginning. In addition, all the resulting UIs are consistent by construction since the transformation engine started from the same task and domain models. For this purpose, UsiXML [13] has been extended to support the expression of these rules and modalities. For each modality, a series of design options exist that allow the designers to change their design according to the context of use. As future work, UsiXML will be extended so as to support the use of other modalities, such as tactile interaction.

6. REFERENCES

[1] Aït-Ameur, Y., Breholée, B., Girard, P., Guittet, L., Jambon, F.: Formal Verification and Validation of Interactive Systems Specifications: From Informal Specifications to Formal Validation. In: R. Jacquart (ed.), Proc. of 18th IFIP World Computer Congress. Kluwer Academics Publishers, Dordrecht (2004) 61–76.

[2] Berti, S., Paternò, F.: Model-based Design of Speech Interfaces. In: Proc. of 10th Int. Conf. on Design, Specification, and Verification of Interactive Systems DSV-IS'2003 (Madeira, 4-6 June 2003). LNCS, Vol. 2844. Springer Verlag, Berlin (2003) 231–244.

[3] Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., and Vanderdonckt, J.: June 2003, A Unifying Reference Framework for Multi-Target User Interfaces. Interacting with Computers 15(3) 289–308.

[4] Chen, M., Luo, J., Dong, S.: Task-Oriented Synergistic Multimodality. In: Proc. of 1st Int. Conf. on Multimodal Interface ICMI'96 (Beijing, October 1996).

[5] Coutaz, J., Nigay, L., Salber, D., Blandford, A., May, J., Young, R.: Four Easy Pieces for Assessing the Usability of Multimodal Interaction: the CARE properties. In: Proc. of 5th IFIP TC 13 Int. Conf. on Human-Computer Interaction Interact'95, 115–120.

[6] Göbel, S., Buchholz, S., Ziegert, T., Schill, A.: Device Independent Representation of Web-based Dialogs and Contents. In: Proc. of the IEEE Youth Forum in Computer Science and Engineering YUFORIC'01 (Valencia, November 2001). IEEE Computer Press.

[7] Hastie, H., Johnston, M., Ehlen, P.: Context-sensitive Help for Multimodal Dialogue. In: Proc. of the 6th ACM Int. Conf. on Multimodal Interfaces ICMI'2004. ACM Press, New York (2004) 93–98.

[8] IBM Voice Toolkit for WebSphere Studio. Int. Business Machines (10 September 2004) http://www-306.ibm.com/software/ pervasive/voice_toolkit/.

[9] IBM Multimodal Browser. Int. Business Machines (10 September 2004) http://www-306.ibm.com/software/pervasive/mul timodal/.

[10] Katsurada, K., Nakamura, Y., Yamada, H., Nitta, T.: XISL: A Language for Describing Multimodal Interaction Scenarios. In: Proc. of 5th Int. Conf. on Multimodal Interfaces ICMI'2003 (Vancouver, 5-7 November 2003). ACM Press, New York (2003) 281–284.

[11] Larson, J.A., Raman, T.V., Raggett, D.: Multimodal Interaction Framework, W3C Note. W3 Consortium (6 May 2003), http://www.w3.org/TR/mmi-framework.

[12] Limbourg, Q., Vanderdonckt, J., Addressing the Mapping Problem in User Interface Design with UsiXML. In: Ph. Palanque, P. Slavik, M. Winckler (eds.), Proc. of TAMODIA'2004 (Prague, November 15-16, 2004). ACM Press, New York (2004) 155–163.

[13] Limbourg, Q., Vanderdonckt, J., Michotte, B., Bouillon, L., Lopez-

Jaquero, V.: UsiXML: a Language Supporting Multi-Path Development of User Interfaces. In: Proc. of EHCI-DSVIS'2004 (Hamburg, 11-13 July 2004). Kluwer Academics, Dordrecht (2005) 207-228.

[14] Limbourg, Q.: Multi-path Development of User Interfaces. Ph.D. thesis. Université catholique de Louvain, Louvain-la-Neuve (2004).

[15] Maes, S.: Position Statement for Multimodal Workshop. In: Proc. of W3C Workshop on Multimodal interaction MMI'2004 (Sophia Antipolis, 19-20 July 2004). W3C (2004) http://www.w3.org/2004/02/mmi-workshop/maes-oracle.pdf.

[16] 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, to appear.

[17] Mueller, W., Schaefer, R., Bleul, S.: Interactive Multimodal User Interfaces for Mobile Devices. In Proc. of 37th Hawaii Int. Conf. on System Sciences HICSS'2004 (Big Island, 5-8 January 2004). IEEE Computer Society Press, Los Alamitos (2004).

[18] Palanque, Ph., Schyn, A., A Model-Based Approach for Engineering Multimodal Interactive. In: Proc. of 9th IFIP TC13 Int. Conf. on Human-Computer Interaction INTERACT'2003 (Zurich, 1-5 September 2003). IOS Press, Amsterdam (2003) 543–550.

[19] Simon, R., Jank, M., Wegscheider, F.: A Generic UIML Vocabulary for Device- and Modality Independent User Interfaces. In: Proc. of the 13th Int. World Wide Web Conference WWW'13 (New York, 17-22 May 2004). ACM Press, New York (2004).

[20] X+V–Authoring, Deploying and Consuming Multimodal Services. Versatile Multimodal Solutions (2004) http://www.sys-con.com/xml/article.cfm?id=615.

[21] Ziegert, T., Lauff, M., Heuser, L.: Device Independent Web Applications - The Author Once - Display Everywhere Approach. Proc. of 4th Int. Conf. on Web Engineering ICWE'04 (Munich, 28-30 July 2004). Lecture Notes in Computer Science, Vol. 3140. Springer-Verlag, Berlin (2004) 244–255.