Towards Evolutionary Design of Graphical User Interfaces

Josefina Guerrero-García[1,2], Juan Manuel González-Calleros[1,2], Jean Vanderdonckt[3]

[1]Universidad Autónoma de Puebla, Facultad de Ciencias Computacionales
Ciudad Universitaria, PC. 72592 , Puebla (México)
[2]R&D Unit, Estrategia 360 (México)
{jguerrero, juan.gonzalez}@cs.buap.mx

[3]Louvain Interaction Laboratory , Louvain School of Management, Université catholique de Louvain
Place des Doyens, 1 – B-1348 Louvain-la-Neuve (Belgium)
Jean.vanderdonckt@uclouvain.be

ABSTRACT

In this paper we argue that user interface design should evolve from iterative to evolutionary design in order to support the user interface development life cycle in a more flexible way. Evolutionary design consists of considering any input that informs to the lifecycle at any level of abstraction and its propagation through inferior and superior levels (vertical engineering) as well as the same level (horizontal engineering). This lifecycle is particularly appropriate when requirements are incomplete, partially unknown, to be discovered progressively. We exemplify this lifecycle by a method for developing user interfaces of workflow information systems. The method involves several models (i.e., task, process, workflow, domain, context of use) and steps. The method applies model-driven engineering to derive concrete user interfaces from a workflow model imported into a workflow management system in order to run the workflow. Instead of completing each model step by step, any model element is either derived from early requirements or collected in the appropriate model before being propagated in the subsequent steps. When more requirements are elicited, any new element is added at the appropriate level, consolidated with the already existing elements, and propagated to the subsequent levels.

Author Keywords

Model Driven Engineering, UsiXML, UIDL, User Interfaces, Evolutionary design, Workflow systems.

General Terms

Design, Experimentation, Human Factors, Verification.

ACM Classification Keywords

D2.2 [**Software Engineering**]: Design Tools and Techniques – *Modules and interfaces; user interfaces*. D2.m [**Software Engineering**]: Miscellaneous – Rapid Prototyping; reusable software. H5.2 [**Information interfaces and presentation**]: User Interfaces – *Prototyping*.

INTRODUCTION

In Software Engineering (SE), *evolutionary prototyping* is a software development lifecycle (SDLC) model in which a software prototype is progressively created for supporting demonstration to customer and for elicitation of system requirements elaboration [11]. Evolutionary prototyping model includes four main phases: 1) definition of the basic requirements, 2) creating the working prototype, 3) verification of the working prototype, and 4) improvement of the requirements elicited.

Evolutionary prototyping model allows creating working software prototypes faster and may be applicable to projects where: system requirements early are not known in advance, new software needs to be created, and developers are not confident enough in existing software development experience.

Evolutionary design extends evolutionary prototyping in the sense that a system prototype, once it has been sufficiently refined and validated, may go further in the software development life cycle: from early to advanced design, if model-driven engineering is used as a method for the user interface (UI) development life cycle.

The remainder of this paper is structured as follows: the next section introduces the concept of evolutionary design, its motivation, and its definition. Next, the full implementation of this concept is applied on a software tool, and exemplified through two case studies. An experiment conducted to determine what the superior factors of evolutionary design over a traditional SDLC are. Last, we provide some related work and our conclusions.

EVOLUTIONARY DESIGN CONCEPT

Evolutionary design consists of considering any input that informs to the lifecycle at any level of abstraction and its propagation through inferior and superior levels (vertical engineering) as well as the same level (horizontal engineering). A propagation is defined by an operation type (i.e., creating, deleting, modifying), a source and a target (e.g., a model entity, a model attribute, a model relationship), and a direction (i.e., forward, backward or bidirectional). For instance, creating a class in one model may propagate the creation of another class in another model. If there is continuous feedback between the evolving design representations, each design activity can benefit from information derived in the other steps. For example, user interface design benefits from task analysis; problems in the task analysis can in turn be revealed during user interface design, allowing benefit to be derived in both directions [1]. We give a

taxonomy of these propagations as model-to-model (M2M) transformations in HCI. A taxonomy of model transformations [16] showed that is important to consider horizontal versus vertical transformations. A horizontal transformation is a transformation where the source and target models reside at the same abstraction level.

A typical example is refactoring. A vertical transformation is a transformation where the source and target models reside at different abstraction levels. A typical example is refinement, where a specification is gradually refined into a full-fledged implementation, by means of successive refinement steps that add more concrete details. The remainder of this paper will focus only on creating operations as a systematic example for discussion propagations. Other operation types are similar.

UI DESIGN METHOD FOR WORKFLOW

In this section, we briefly outline the steps of a method for designing GUIs of workflow information systems (WfIS). Since this paper is aimed at addressing the problem of propagation in evolutionary design, this outline is explained here only with a level of details that is useful for the rest of the discussion and for the sake of evolutionary design. For more details, please refer to [8]. We then provide a series of propagations that have been implemented in FlowiXML, a software that support evolutionary design based on the aforementioned method. The method is composed on the following major steps: Workflow information system requirements, design and implementation.

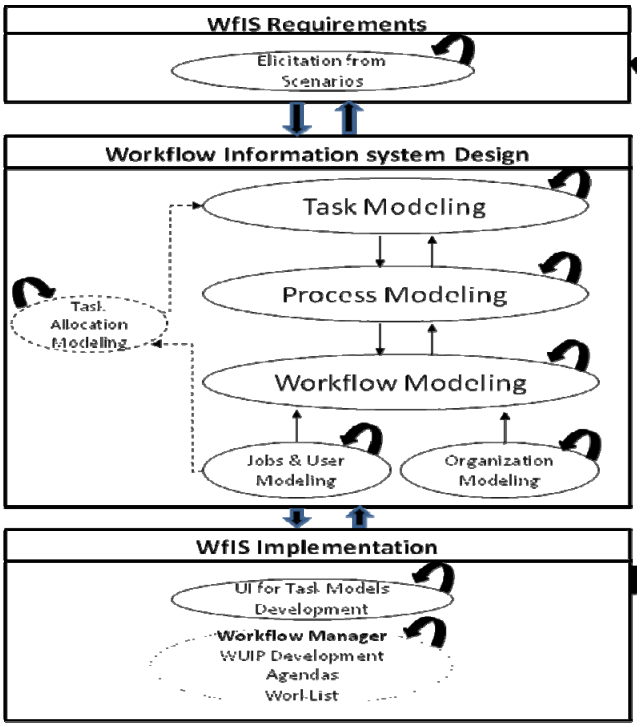

Figure 1. Methodology to develop workflow information systems

In Figure 1, forward and backward arrows denote the propagation of information from one model to another. For instance, a new task model must make available a task for a process model and vice versa, a new task in a process model might be detailed with a task model. Jobs, user stereotypes and organizational modeling just affect the workflow model. Then the workflow model makes them available for process modeling and task modeling. This particular aspect of concepts propagation was significantly useful for the software tools that support FlowiXML methodology [10].

"The process is iterative; once the implementation is done the system can be refined starting from the requirements or the design". The modeling activities also are iterative. It is important to mention, the evaluation of the final implemented system is not in the scope of this paper. As many methods exist for this purpose and depending of the context of use they are used, we just assumed that in some way the system will be tested and if there is a need to iterate then there is a point back into the requirements definition and the design of the workflow information systems.

Identification Criteria

One important and recurrent element that is of our interest is the concept of "task". There are fundamental works that define workflow, processes, and tasks [23]. We established [5] a set of criteria to identify the concepts of: task, process and workflow. We looked at four dimensions surrounding the task execution (i.e., time, space, resources, and information). Any variation of any of these four dimensions, taken alone or combined, thus generates a potential identification of a new concept. At first everything is classified as tasks. A task could be part of a process model or a task model. Existing knowledge on task identification criteria is again relevant to make such separation. Table 1 shows part of these identification criteria.

	Time	Space (location)	User Stereotype
Workflow	Series of time periods	Different locations	Different groups of resources
Process	Series of time periods	Same location	One resource or a group of resources
Task	Same time period	Same location	Same resource

Table 2. Identification criteria.

A task model is composed of tasks performed by the same resource, in the same location and in the same time period. The reader must not be confused by the fact that during the execution of the task one of these properties might be changed. For instance, the task "insert client information" could be suspended during the execution of the task, for instance, while answering the phone, consequently when the user resumes the execution of the task the time period changes, it is no longer the same. However, this does not affect the nature of the task and its time periodicity; these kind of external events are part of the task life cycle.

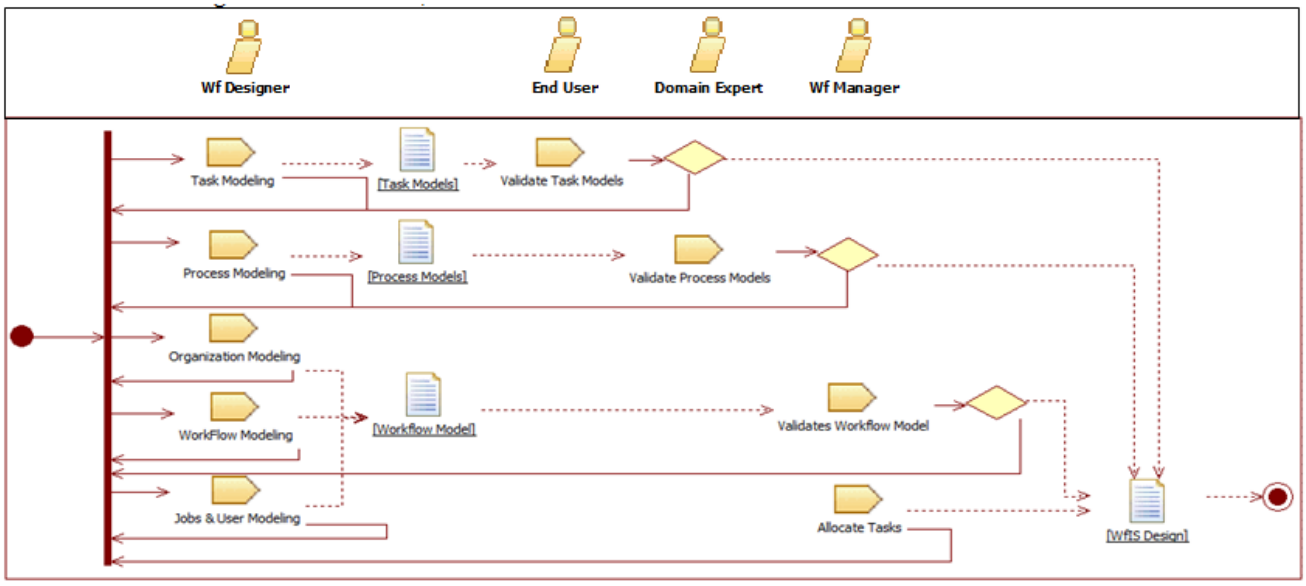

Figure 2. Evolutionary design of graphical user interfaces activity diagram.

Location and change in user is something evident to identify but the time periodicity might be tricky to discover. If the designer confronts difficulties we propose the use of a timeline to gather tasks just to be sure that they are executed in the same time series. A process model is composed of tasks performed by one resource, or one or more group of resources; the location is the same; the time periodicity changes. Finally a workflow is composed of processes performed by one or more group of resources; located in different organizational units (within the same or different organizations); and the time periodicity of the processes changes.

EVOLUTIONARY DESIGN OF GUI

Model-based user interface development environments (MB-IDEs) [22] seek to describe the functionality of a user interface using a collection of declarative models. The WfIS design is composed of several models (workflow, process and task) and several actors are involved (Wf designer, end-user, domain expert and Wf manager). The first activity corresponds to the consolidation of the concepts identified in the elicitation of the scenario. In [13] we have investigated three different techniques for eliciting model elements from fragments found in a textual scenario in order to support activities of scenario-based design. The design process is depicted in Figure 2, the drawings used corresponds to the Software Process Engineering Meta-model (SPEM) notation, promoted by the OMG.

The system design is an activity that can start from any model (Figure 2) except for the task allocation because it needs tasks and resources already defined. The design of a workflow permits designers to identify concepts freely and to start to detail based on their preferences. One designer may prefer to get into details of task modeling before describing a process model. Once the task models are ready, then it can model the processes that then are arranged to represent the workflow. Another designer might have a better understanding of the problem with the workflow model (more abstract view of the problem) and then start to refine by adding process models and finished with task models. There is no constraint on the starting and end point for modeling, just to be sure keep the traceability of the concepts that are shared in different models (task model is part of process and a process model is part of a workflow model). To support traceability, the transformation language or tool needs to provide mechanisms to maintain an explicit link between the source and target models of a model transformation [15].

The end-user is the responsible of validating task models. Note by end-user we understand the person who actually is in charge of performing this task, i.e., the most qualified to say something about it. The process is validated by the domain expert. This is due to the fact that a process requires a higher level of understanding of the problem. In this view it could be any person in the organization that through the requirements analysis has been identified to be most familiar with the whole process modeled.

Finally the workflow model is validated by the Wf manager, who is the person or group of persons with an understanding of the whole workflow when processes are grouped. It is also, the Wf manager who is in charge of allocating tasks in a process to resources. The final result is the Workflow Information Systems (WIS) design. All these design activities are also accompanied with guidance for the modeling activities.

Progressing from one model to another involves transforming the model and mapping pieces of information contained in the source model onto the target model [3].

SYSTEM IMPLEMENTATION

The implementation of the system refers to the UI generation of tasks. There is a plethora of user interface description languages that are widely used, with different goals and different strengths. A review of XML user interface description languages was produced [9] that compares a significant selection of various languages addressing different goals, such as multi-platform user interfaces, device-independence, and content delivery.

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

RIML semantics is enhanced to cover pagination and layout directives in case pagination needs to be done, in this sense it was possible to specify how to display a sequence of elements of the UI. RIML is device independent and can be mapped into a XHTML specification according to the target device.

No concepts or task models are explicitly used in SeescoaXML. The entry point of this forward engineering approach is therefore located at the level of Abstract UIs.

SunML presents a reduced set of elements that seems to be not enough, but the composition of widgets is used to specify more complex widgets. In TeresaXML the whole process is defined for design time and not for run-time. At the AUI level, the tool provides designers with some assistance in refining the specifications for the different computing platforms considered. The AUI is described in terms of Abstract Interaction Objects (AIOs) that are in turn transformed into Concrete Interaction Objects (CIOs) once a specific target has been selected.

UIML is a meta-language, is modality-independent, target platform-independent and target language-independent. The specification of a UI is done through a toolkit vocabulary that specifies a set of classes of parts and properties of the classes. Different groups of people can define different vocabularies: one group might define a vocabulary whose classes have a 1-to-1 correspondence to UI widgets in a particular language, whereas another group might define a vocabulary whose classes match abstractions used by a UI designer. UsiXML is structured according to different levels of abstraction relying on a transformational approach to move among them. It ensures the independence of modality thanks to the AUI level. It is supported by a collection of tools that allow processing its format.

WSXL is a Web services centric component model for interactive Web applications. WSXL applications consist of one or more data and presentation components, together with a controller component which binds them together and specifies their interrelated behavior.

XICL is a language to UI development by specifying its structure and behavior in an abstract level than using only DHTML. It also promotes reuse and extensibility of user interface components. The developer can create new and more abstract UI components. XIML supports design, operation, organization, and evaluation functions; it is able to relate the abstract and concrete data elements of an interface; and it enables knowledge-based systems to exploit the captured data.

Even that the model describe above can be incorporated to UIML, XICL or XIML languages, we have considered that UsiXML is the suitable language that could accommodate workflow concepts in a flexible way because its documentation is available including its meta-models and deep analysis can be done, it has a unique underlying abstract formalism represented under the form of a graph-based syntax, UsiXML allows reusing parts of previously specified UIs in order to develop new applications, It is supported by a collection of tools that allow processing its format, it allows cross-toolkit development of interactive application thanks to its common UI description format.

We rely on UI generation from task models techniques [14,17,18] for deriving UI. Some techniques apply task models, among other models, as a source to develop UIs, thus we model the allocation patterns with task models to derive Workflow User Interface Patterns (WUIPs). Finally we identify potential information relevant for the implementation of the workflow manager with the UI flow. After having defining the UIs involved in the workflow, we need now to link all the UIs: the one for the workflow manager and the ones for the workflow tasks. This will be achieved thanks to the user interface flow.

During the execution of work, information passes from one resource to another as tasks are finished or delegated; in FlowiXML we use an agenda assigned to each resource to manage the tasks that are allocated/offered to him. The manager uses the work list to view and manage tasks that are assigned to each resource. By linking UIs we expect to solve the problem of synchronizing the communication among them. We introduce some rules that can be applied to facilitate the modeling of the UIs flow that is relevant for the implementation of a WfIS.

Software Support

The software support allows modeling the general WfIS defining workflow, process and task models, organizational units, jobs and resources involved, and allocation of task to resources. The software was developed based on the conceptual modeling presented in [8]. It is the result of constant improvements based on informal evaluations, interviews with the users after modeling a WfIS. The family of software tools is shown in Figure 3; we will explain them along with the description of some case studies. A video on YouTube illustrates the usage of the software tools (http://www.youtube.com/user/FlowiXML).

APPLYING THE METHODOLOGY ON REAL LIFE CASE STUDIES

We have worked with several case studies applying our methodology. As the method does not force any sequence of steps, all case studies followed different development paths. This shows the flexibility of the proposed method.

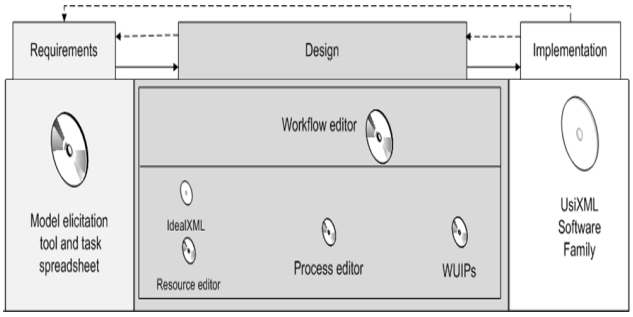

Figure 3. Software tool.

To gather the case studies we asked some students to propose a real life case study with at least twenty complex tasks developed in four different organizational units, involving sufficiently diversity of resources. The students, under our supervision, developed the case studies by completing a report including: introduction to the problem; scenario where the problem takes place; working hypotheses; task identification; task modeling; organizational modelling; jobs and user modeling; process modeling; workflow modelling; UI definition; workflow simulation and implementation; workflow analysis.

Students were free to select IdealXML or CTTE as the tool for task modeling. We will use one case study to exemplify the methodology and the evolutionary design, the reader could find the other case studies published in FlowiXML website (http://www.usixml.org/index.php?mod=pages& id=40).

The first task is about of how to extract WfIS concepts from textual scenarios describing the problem to be solved. The problem related to a library management, the whole process that makes new books and existing books available to borrow. A series of tasks have been identified: Insert book, correct notice, find mistakes, publish notice, insert notices, find book, among others. Once all tasks have been selected, grouped and described, a task model is constructed accordingly to its order.

Tasks are gathered in their corresponding organizational units and the selection and definition of jobs and users for the corresponding tasks are graphically selected (Figure 4). The next step consists on the definition of a process which indicates which tasks must be performed and in what order. Thus answering the question: what to do? After having identified tasks that are part of a process, then they have to be related to each other by means of process operators.

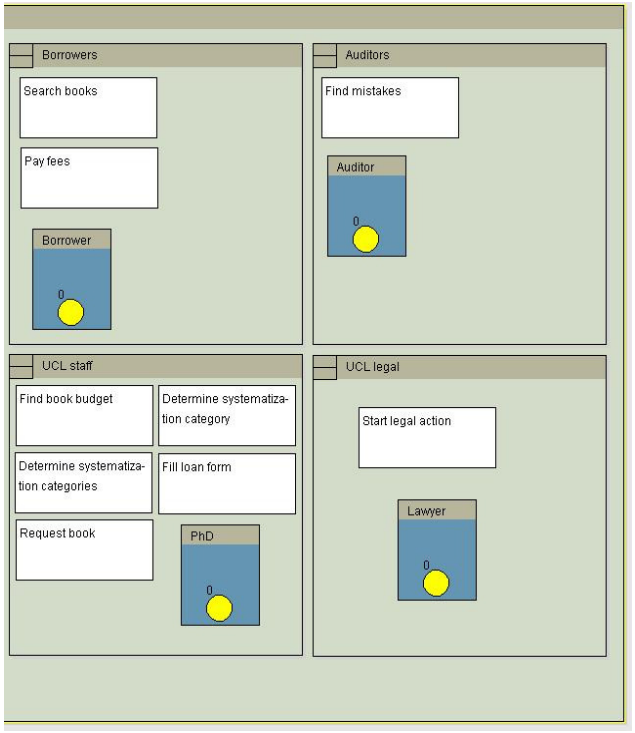

Figure 4. Organizational modeling and resource allocation.

Three concepts are relevant for a process model: place (process state), transition (task), and process operators, this is known as a WF-net [23]. In Figure 5 the workflow of the case study is represented.

Task models do not impose any particular implementation so that user tasks can be better analyzed without implementation constraints. This kind of analysis is made possible because user tasks are considered from the point of view of the users need for the application and not how to represent the user activity with a particular system. For instance, let's consider the *search notice* task; the notice for a given book (if exists) can be searched using different parameters such as: ISBN, editor, keywords, title, authors, publication year; then the user validates its search information and launch the search; the system convey the results of the search; finally the user selects the code of the desired book. Figure 6 shows the task modeling using the CTT graphical notation [17].

We rely on UI generation from task models techniques [14,17,18] for deriving UI from task models, thus the whole GUI is developed thanks to this methodology. For instance the GUI for the Check in task is shown in figure 7.

Finally, it is worth to say that the workflow modeling allows the users to identify portions of the workflow where some optimization could be introduced.

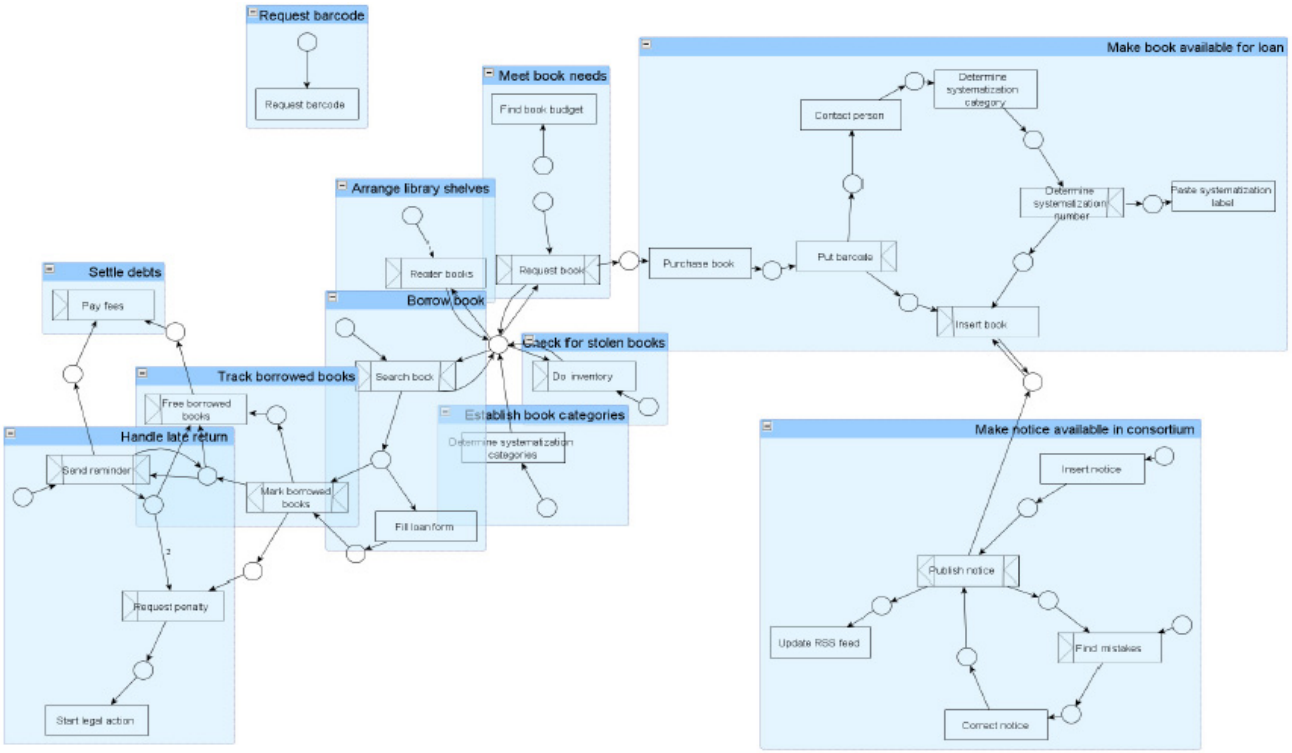

Figure 5. Workflow modeling for the library management system.

In our example, the current workflow could be improved by adding some patterns to optimize the workflow by minimizing the number of possible freezes. Moreover we added some flexibility to the workflow by allowing several resources to have the same job where it is not actually the case (such as for the librarian job which has only one resource associated currently).

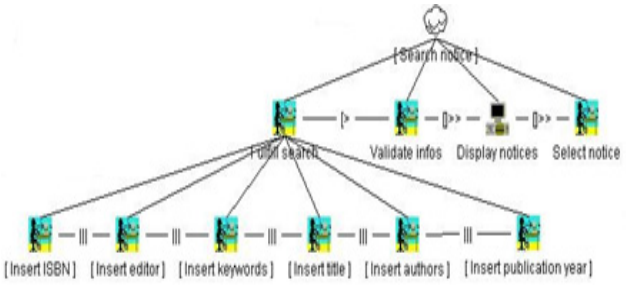

Figure 6. Task model.

Another case study is SPORT-KIT project. SPORT-KIT is designed to be a training system like the Wii Fit® but more evolved and designed for elderly people. A battery of tests is run automatically with the users' interactions to know his/her wellness and physical capabilities so that we can train them effectively to improve them. Also, SPORT-KIT involves a tactile interface, a board with pressure sensor and a webcam. Following the identification criteria, we identify 28 tasks: Start SPORT-KIT, Start test, Personal test, Ask size, Ask waist measurement, Ask weight Compute BMI, Start EUROQUOL Test, 5 questions of the test, Scale test, Store EUROQUOL result, Start STAND test, STAND test 2 feet, STAND test 1 feet left, STAND test 1 feet right, STAND test 1 feet eye closed, Store STAND test result, Start STEP Test, STEP Test, Store STEP Test result, Start FORCE Test, FORCE Test, Store FORCE Test result, Start SOUPLENESS Test, SOUPLENESS Test, Store SOUPLENESS Test result, Compare weight, Compute training program. Some of them are shown in Table 2. As we can observe, the goal of the SPORT-KIT system is to compute a training program.

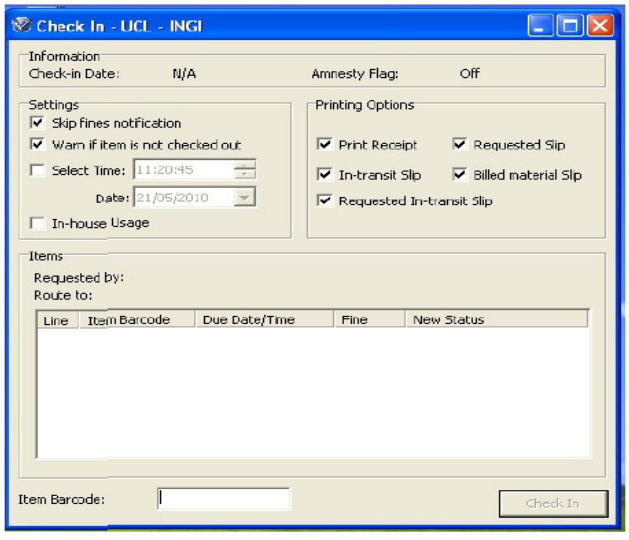

Figure 7. GUI for the *Check in* task.

We consider only the project **SPORT-KIT**, because the subject is deep enough with this single aspect of the final product. Here is the list of organizational units involved in this part:

- *Project leader*. Gives the interface and architecture to programmers, controls the UI and interaction.
- *Programmers*. Implement the solution; work with foundation server to share code amongst the project progression.
- *Testers*. Test the software as it's now, track bugs, give input on usability of the HMI.
- *Final testers*. Test the final software, give feedbacks and input to project leader and programmers.
- *Physical trainer*. Makes the training program, chooses the tests and helps to implement; reports to project leader.

ID	Task name	Definition	Nature
1	Start SPORT-KIT	User starts the program.	Interactive
2	Start Test	No previous test was done by the user	Automatic
3	Personal Test	User introduce information about himself	Automatic
4	Ask Size	User introduce his size in meter	Interactive
5	Ask waist measurement	User introduce his waist measurement, if he doesn't know clothes size is proposed	Interactive
6	Ask weight	User introduce his weight	Interactive
7	Compute BMI	Compute the body mass index and store it in the database	Automatic
8	Start EUROQUOL Test	Explain to user what he has to do with this test: just touch the best answer	Automatic
9	5 questions of the test	Ask the question about health status of the person	Interactive
10	Scale Test	The user has to evaluate his health status on a scale from 0 to 100	Interactive

Table 3. Some identified tasks.

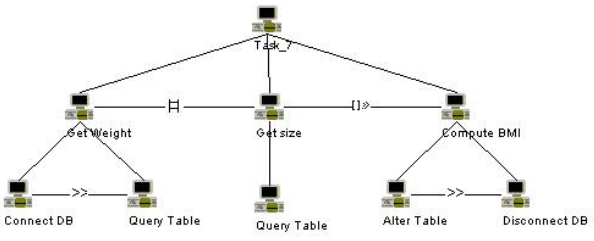

Figure 8. Compute BMI task model.

For each task we can have a task model following the CTT notation [17]. For instance, Figure 8 shows the subtasks needed in order to compute the body mass index and store it in the database (task 7). After, we can identify which jobs are in charge of a particular task; in most of the task the principal actor is the user, the helper collaborate in some of them with the user, and the physician participates just in 5 tasks. User participates in the following tasks: Start Sport, Start Test, Personal Test, Ask Size, Ask waist measurement, Ask weight, Compute BMI, Start EUROQUOL Test, 5 Questions of the test, Scale Test, Store EUROQUOL result, Start STAND test, STAND test 2 feet, STAND test 1 feet left, STAND test 1 feet right, STAND test 1 feet eye closed, Start STEP Test, STEP Test, Start FORCE Test, FORCE Test, and SOUPLENESS Test. Helper participates in the following tasks: Start STAND test, STAND test 2 feet, STAND test 1 feet left, STAND test 1 feet right, STAND test 1 feet eye closed, STEP Test, and Start SOUPLENESS Test. Physician participates in the following tasks: Store STAND test result, Store STEP Test result, Store FORCE Test result, Store SOUPLENESS Test result, and Compare weight.

Next step in this case study is producing the process model. The identification criterion was the ability to regroup tasks in a consistent sub-process of the whole process. Also, consistency was automatically checked by the software we used, so we cannot break the rules.

The frontiers of the processes are defined as follows:

Process name	Tasks
Clique sur SPORT-KIT	1, 2
Test **papier**	3 → 10
Test **pression**	21, 22
Test **back scratch**	24, 25
Calcul IMC	7
Test **step**	18, 19
Test **équilibre**	12 → 16
Sauve résultats dans la BD	11, 17, 20, 23, 26, 27

The complete workflow that uses the previously identified processes is shown in Figure 10, we use Petri net to model the workflow.

Finally, UIs for the case study are derived using the model-driven approach of UsiXML, a set of *transformation rules* were applied [see www.usixml.org for more information]. It is not the scope of this paper to address UI development but relies on existing work. UsiXML model-driven UI development uses as input a task model, enriched with our meta-description, and transform it into a user interface.

Importance of the Evolutionary Design

Adding/modifying a task, user or object attribute in the textual scenario results into creating/modifying an attribute in the corresponding model. Similarly adding/modifying a new task on a workflow model affects the task modeling, as

another task must find its place in an existing task model or a new task model must be designed.

The propagation of these information impacts the different layers of the development process, without information traceability the evolutionary design would not be possible. In Figure 9 the methodological path editor is shown, where the different properties to use during the development process.

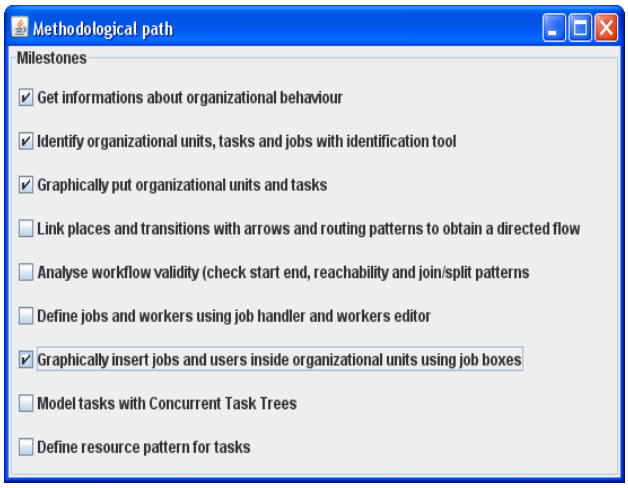

Figure 9. Methodological path editor.

For instance, Milestones are used to get a methodological path reminder. The workflow editor user may model the workflow using many different step orderings. It is a practical way to note what you still have to do, telling you where you are (after a holiday break for example).

RELATED WORK

The multidisciplinary nature of system design can lead to a lot of different works in the literature which will allow developers to validate the quality of the system development in an efficient manner. In [20] some concepts for evolutionary systems and some of the technology investigations that are felt to support those concepts are presented. It discusses the evolutionary design of complex software which objective is to provide economic methods for systems to keep up with changer requirements over their lifetimes by providing a strong information base for evolution, enabling analysis of impacts of intended changes, enabling design and implementation of more adaptable systems.

The work of Chong Lee [2] draws from extreme programming (XP), an agile software development process, and claims-centric scenario-based design (SBD), a usability engineering process. Extreme programming, one of the most widely practiced agile methodologies, eschews large up-front requirements and design processes in favor of an incremental, evolutionary process. In [1] we can see how different user interface design artifacts can be linked to expose their common elements, facilitating the communication on which coevolutionary design is based.

The Clock architecture language [6] was designed to support the evolutionary design of architectures for interactive, multi-user systems. In [24] authors discuss an approach for linking GUI specifications to more abstract dialogue models and supporting an evolutionary design process, which is supported by patterns. Additionally, a proposal is presented of how to keep connections between concrete user interface, abstract user interface and a task model. Based on evolutionary character of software systems, Rich and Waters [19] have developed an interactive computer aided-design tool for software engineering; it draws a distinction between algorithms and systems, centering its attention on support for the system designer. In [4] authors have argued that task and architecture models can be semi-automatically linked.

They have developed a computer-based tool, *Adligo*, which generates links between the UAN as a task model and the Clock architecture as an architecture model. To support co-evolutionary software development the linkage process between the models can be carried out at any stage in the development cycle; none of the design artifacts has even to be complete. Krabbel *et al*. [12] adopted a variation of object-orientedness for constructing application software as a product and combined it with an evolutionary development strategy based on prototyping. The approach focuses on a close relationship between the tasks and concepts of the application domain and the software model. In [21] an approach to developing systems which can be summarised as 'analyse top-down, design middle-out, and build bottom-up' is presented. The novelty of the approach lies in its use of task analysis to define an appropriate domain for the system and then the use of a working prototype to grow a system from the bottom up.

Model-based user interface development environments show promise for improving the productivity of user interface developers and possibly for improving the quality of developed interfaces. There are many MB-UIDEs that follow a formalized method, but their supporting tools do not provide facilities to change the sequence of the method activities, thus restricting the possibilities to adapt the method. The TEALLACH design process [7] aims to support the flexibility for the designer lacking in existing environments by providing a variety of routes in the process; from one entry point, the designer/developer can select any model to design independently or associate with other models.

CONCLUSION AND FUTURE WORK

Many existing software engineering processes are unable to account for continuous requirements and system changes requested throughout the development process. Evolutionary design has emerged to address this shortcoming. In this paper, a methodology to support evolutionary user interface development life cycle in a more flexible way was introduced. The methodology supports the evolutionary lifecycle including any level of abstraction to start the design.

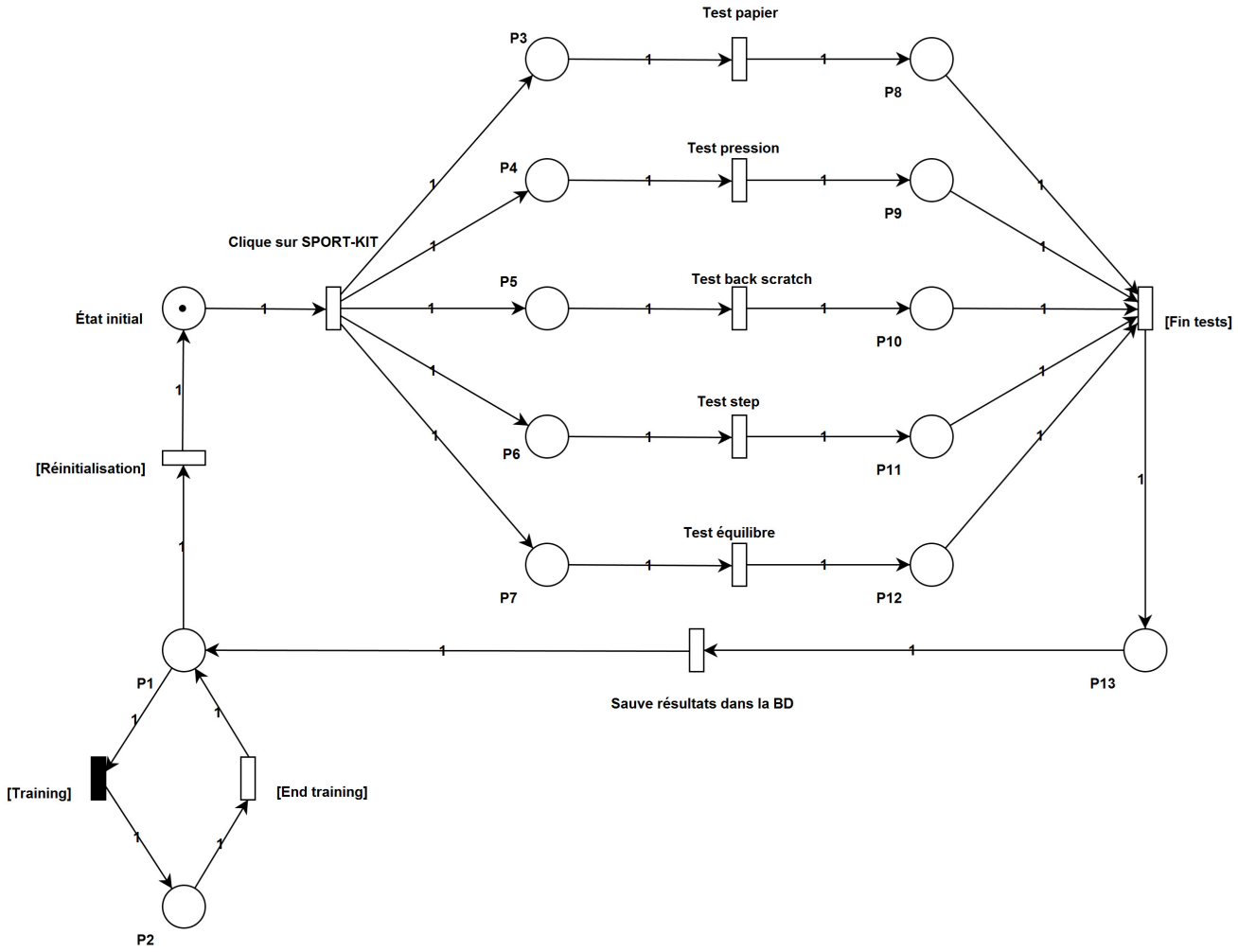

Figure 10. Partial view of the workflow for the SPORT-KIT project.

It is widely accepted that evolutionary design is a natural and preferred design process. A workflow editor was developed to support the method proposed in this paper. A collection of case studies is gathered to validate the methodology. As future work we plan to consider to extent our work to support the development of Post-wimp UIs. Also we will apply the methodology to different context of use, such as: education, training, medicine.

ACKNOWLEDGMENTS

We acknowledge the support of the ITEA2 Call 3 UsiXML (User Interface extensible Markup Language – http://www.usixml.org) European project under reference #2008026 and its support by Région Wallonne DGO6, and the support of the Repatriation program by CONACYT. We thank all UIDL reviewers for their fruitful feedback that contribute in the enhancement of this paper.

REFERENCES

1. Brown, J., Graham, N., and Wright, T. The Vista Environment for the Coevolutionary Design of User Interfaces. In *Proc. of CHI'98*. ACM Press, New York (1998), pp. 376–383.
2. Chong Lee, J. Embracing agile development of usable software systems. In *Proc. of Extended Abstracts of CHI'2006*. ACM Press, New York (2006), pp. 1767–1770.
3. Clerckx, T., Luyten, K, and Coninx, K. The Mapping Problem Back and Forth: Customizing Dynamic Models while preserving Consistency. In *Proc. of TAMODIA'-2004*. ACM Press, New York (2004), pp. 33–42.
4. Elnaffar, S. and Graham, N.C. Semi-Automated Linking of User Interface Design Artifacts. In *Proc. of CADUI'-99*. Kluwer Academic Pub. (1999), pp. 127–138.
5. Garcia, J.G., Vanderdonckt, J., and Lemaigre, Ch. Identification Criteria in Task Modeling. In *Proc. of 1st IFIP TC 13 Human-Computer Interaction Symp. HCIS'2008*. IFIP, Vol. 272, Springer, Boston (2008), pp. 7–20.
6. Graham, T.C.N. and Urnes, T. Linguistic support for the evolutionary design of software architectures. In *Proc. of ICSE'96,* IEEE Computer Society, Los Alamitos (May 1996), pp. 418–427.

7. Griffiths, T., Barclay, P.J., Paton, N.W., McKirdy, J., Kennedy, J., Gray, P.D., Cooper, R., Goble, C.A., and Pinheiro, P. Teallach: a Model-based User Interface Development Environment for Object Databases. *Interacting with Computers 14*, 1 (December 2001), pp. 31–68.
8. Guerrero García, J., Lemaigre, Ch., González Calleros, J.M., and Vanderdonckt, J. Towards a Model-Based User Interface Development for Workflow Information Systems. *Int. J. of Universal Comp. Science 14*, 19 (Dec. 2008), pp. 3236-3249.
9. Guerrero-García, J., González-Calleros, J.M., Vanderdonckt, J., and Muñoz-Arteaga, J. A Theoretical Survey of User Interface Description Languages: Preliminary Results. In *Proc. of LA-Web/CLIHC'2009* (Merida, November 9-11, 2009), IEEE Computer Society Press, Los Alamitos, 2009, pp. 36-43.
10. Guerrero García, J., A Methodology for Developing User Interfaces to Workflow Information Systems. PhD. Thesis, UCL (2010).
11. Hekmatpour, S. Experience with evolutionary prototyping in a large software project. *ACM SIGSOFT Software Engineering Notes 12*, 1 (Jan. 1987), pp. 38–41.
12. Krabbel, A., Wetzel, I., and Züllighoven, H. On the inevitable intertwining of analysis and design: developing systems for complex cooperations. In *Proc. of DIS'97*. ACM Press, New York (1997), pp. 205–213.
13. Lemaigre, Ch., Garcia, J.G., and Vanderdonckt, J. Interface Model Elicitation from Textual Scenarios. In *Proc. of 1st IFIP TC 13 Human-Computer Interaction Symposium HCIS'2008*. IFIP, Vol. 272, Springer, Boston (2008) pp. 53–66.
14. Limbourg, Q. and Vanderdonckt, J. Addressing the Mapping Problem in User Interface Design with UsiXML. In *Proc. of TAMODIA'2004*, ACM Press, New York (2004), pp. 155–163.
15. Loia, V., Staiano, A., Tagliaferri, R., and Sessa, S. An evolutionary hybrid approach to the design of a decision support system. In *Proc. of SAC'00*, Vol. 1, ACM Press, New York (March 2000), pp. 524–528.
16. Mens, T. and Van Gorp, P. A Taxonomy of Model Transformation. *Electronic Notes in Theoretical Computer Science 152* (2006), pp. 125–142.
17. Paternò, F. *Model-Based Design and Evaluation of Interactive Applications*. Springer-Verlag, London (1999).
18. Puerta, A.R. and Eisenstein, J. Towards a general computational framework for model-based interface development systems. *Knowledge-Based Systems 12*, 8 (1999), pp. 433–442.
19. Rich, Ch. and Waters, R.C. Computer aided evolutionary design for software engineering. *ACM SIGART Bulletin 76* (April 1981), pp. 14–15.
20. Salasin, J. and Shrobe, H. Evolutionary design of complex software (EDCS). *ACM SIGSOFT Software Engineering Notes 20*, 5 (December 1995), pp. 18–22.
21. Seaton, P. and Stewart, T. Evolving task oriented systems. In *Proc. of CHI'92*, ACM Press (1992), pp. 463–469.
22. Szekely, P., Retrospective and Challenges for Model-Based Interface Development. In *Proc. of DSVIS'96*, F. Bodart and J. Vanderdonckt (Eds.), Springer-Verlag, 1996, pp. 1-27.
23. van der Aalst, W. & van Hee, K., Workflow Management: Models, Methods, and Systems. THE MIT Press, Cambridge. 2002.
24. Wolff, A., Forbrig, P., Dittmar, A., and Reichart, D. Linking GUI elements to tasks: supporting an evolutionary design process. In *Proc. of TAMODIA'2005*. Springer, Berlin (2005), pp. 27–34.