Revista Română de Interacțiune Om-Calculator 6 (3) 2013, 269-290

Sustaining Designers' and Users' Quality of Life in the Paradigm of Plastic UIs

Eric Ceret[1], Alfonso García Frey[2], Sophie Dupuy-Chessa[3], Gaëlle Calvary[1]

1Grenoble INP, 2UJF, 3UPMF, 123CNRS, 123LIG
41 rue des mathématiques, 38400 Saint Martin d'Hères, France
E-mail: {alfonso.garcia-frey, eric.ceret, sophie.dupuy, gaelle.calvary}@imag.fr

Abstract. Modern User Interfaces need to dynamically adapt to their context of use, i.e. mainly to the changes that occur in the environment or in the platform. Model-Driven Engineering offers powerful solutions to handle the design and the implementation of such User Interfaces. However, this approach requires the creation of an important amount of models and transformations, each of them in turn requiring specific knowledge and competencies. This leads to the need of adapted process models and tools sustaining the designers' work. Moreover, automatic adaptation to new devices implies that users could have questions about the interaction with the same UI in such new devices. As this adaption is automatically performed at runtime, designers cannot foresee all the possible combinations of contexts of use at design time in order to conveniently support the users. For this reason, dynamic help systems are necessary to generate dynamic explanations to the end-user at runtime. This paper presents (1) a new vision of process model flexibility that makes it possible to adapt the process model to the designer's knowledge and know-how, (2) the "flexibilization" of the UsiXML methodology, (3) the principles supporting self-explanatory UIs and (4) the integration of all these notions in UsiComp, an integrated and open framework for designing and executing plastic User Interfaces. UsiComp relies on a service-based architecture. It offers two modules, for design and execution. The implementation has been made using OSGi services offering dynamic possibilities for using and extending the tool. This paper describes the architecture and shows the extension capacities of the framework through two running examples.

Keywords: UI Plasticity, Self-Explanatory Interfaces, Process Model Flexibility, Model-Driven Engineering; User Interfaces; Design Tools.

1. Problem and motivation

With the increasing amount of platforms and devices as well as of the new expectations of users, designers need to create User Interfaces (UIs) that are able to adapt to their context of use, i.e. to the changes that occur in the environment, the platform and/or the user profile. However, the huge

amount of possible combinations of these context elements makes it no longer possible to anticipate and predefine all the eventual situations at design time. Systems have to be designed to be able to adapt themselves to their context of use while preserving usability (Calvary *et al.*, 2001). Model Driven Engineering (MDE), which is based on the generation of applications from models, provides powerful solutions for the creation of such UIs. In this paradigm the models represent the different facets of the system to be created. These models are successively transformed and combined to finally generate the code. This opens possibilities like easier evolutions and reuse (Hamid *et al.*, 2008), dynamic adaptation to the context of use, greater quality, early detection of defects and inclusion of knowledge in executable models (Mohagheghi *et al.*, 2009).

However, creating all the models and all the transformations for an application is a long and complicated work: the designer has to understand the underlying meta-models, write the models that conform to these meta-models and elaborate some transformations. Then, the designer needs to create a system that runs the transformations and generates the final code. The threshold of use, as defined by (Resnick *et al.*, 2005), is high.

Moreover, plastic UIs demand dynamic help systems as well because in plastic UIs the resulting UI is not predefined. UIs may adapt themselves to unforeseen contexts and thus, developers cannot consider all the different contexts of use one by one at design time, becoming complicated to create an efficient and contextual helping system for the end-user. For instance, parts of the UI may be or not be present at a moment or can be distributed on another device where tasks are accomplished in a very different way, for instance, using different modalities. While it is interesting to present help about the absent parts of the UI, it would for instance be very interesting to provide help about how reaching these parts of the UI in an automatic way, or explain to the end users how to accomplish a task in the new device of the new context of use. As this information cannot be specified at design time, means for dynamically compute such explanations are necessary.

This paper presents how the UsiXML process model has been flexibilized to take into account the designers' and developers' skills so that to provide them with adapted guidance. This "flexibilization" relies on a new four-dimensional definition of the process model flexibility (Céret et al., 2013b) and on M2Flex, a flexible process metamodel (Céret et al.,

2013a). This paper also presents UsiComp, a tool for creating a complete set of models (Tasks, Abstract UI, Concrete UI, Domain, Context, Mapping, Quality) and simplifying the creation of transformations. UsiComp design and execution modules, relying on an extensible service-based architecture, include an easy graphical interface that offers an efficient way of creating models by drawing them or by combining predefined components, permitting fast prototyping possibilities. Finally, the paper also presents how this framework integrates a model-driven approach for the generation of self-explanatory UIs, i.e., UIs having the ability for the generating explanations that support the user in the interaction. This support is provided in the form of questions and answers.

This makes it a powerful and innovative tool for designing a system with a flexible MDE with self-explanation abilities that will help the user during the interaction.

2. Quality for the Designer: Process Model Flexibility

Designers and developers are poorly satisfied by methods (Garzotto and Perrone, 2007; Barry and Lang, 2001; Fitzgerald, 1998). They report that methods (1) do not address various kinds of projects and customers' constraints, (2) are difficult to learn and to use, (3) impose complex, linear and rigid processes that are not described in adapted languages.

The authors of the studies conclude that the process models of the methods are not flexible or adaptable enough. According to (Booch, 1993) and (Harmsen, 1997), the process model is part of a method, with the product model and a collection of tools. It focuses on a facet of the design and development process - e.g. the tasks to be completed, the products to be built or the decisions to be made - to describe the activities to be realized.

2.1 Flexibility

Many researches (Basili and Rombach, 1987; Potts, 1989; Harmsen et al., 1994; Bendraou *et al.*, 2007; Hug et al., 2008) have been driven to evaluate process model flexibility. In particular, Harmsen, Brinkkemper and Oei (Harmsen *et al.*, 1994) defined a one-dimension classification for measuring it, ranging from rigid models to the modular construction of process models.

In (Céret *et al.*, 2013b), we proposed a taxonomy for evaluating and comparing process models, based on the study of 49 of them and on several previous works. This taxonomy offers a new definition of flexibility, based on four dimensions: variability, distensibility, completeness and granularability.

Variability is the possibility offered by a process model to designers of making choices in a set of variants. For instance, the goal "create the concept model" can be achieved by several variants like the creation of a UML class diagram or the definition of tables in a database management system.

Granularability is the ability of a process model to support elements with different granularities, e.g. various quantities of details and also to support various languages. For instance, if the process model includes an activity for defining a User Interaction (UI) mockup, an expert UI designer will not need more information. However, a novice UI designer might need either more details or different a different description, dedicated to someone who is not familiar with the specific vocabulary of this domain. A granular process model offers refined and rephrased elements.

Completeness is the possibility of fulfilling or not the proposed process, some activities and/or artifacts are then optional or can be replaced by a predefined result or product. For instance, in a UI design, the activity "define the platforms model" can be optional; when it is not selected, the UI is then designed for an implicit platform, or when several platforms are addressed, it can be replaced by "default" models that the designer picks up in a repository proposed by the process model.

Distensibility is the ability of a process model to be extended or reduced at enactment time, i.e. to accept that proposed elements (such as activities, roles or artifacts) can be avoided from the process or that unexpected elements can be added to it. The issue is here the definition of mechanisms for distending the process model *during its enactment*.

In the following, we present M2Flex, a metamodel based on our taxonomy and we introduce a flexible version of the UsiXML methodology.

2.2 M2Flex, a flexible process metamodel

Figure 1[8] presents an overview of our process metamodel, M2Flex. Hereafter, we detail the main packages that address flexibility. As we focus here on flexibility, all classes and attributes are not extensively presented.

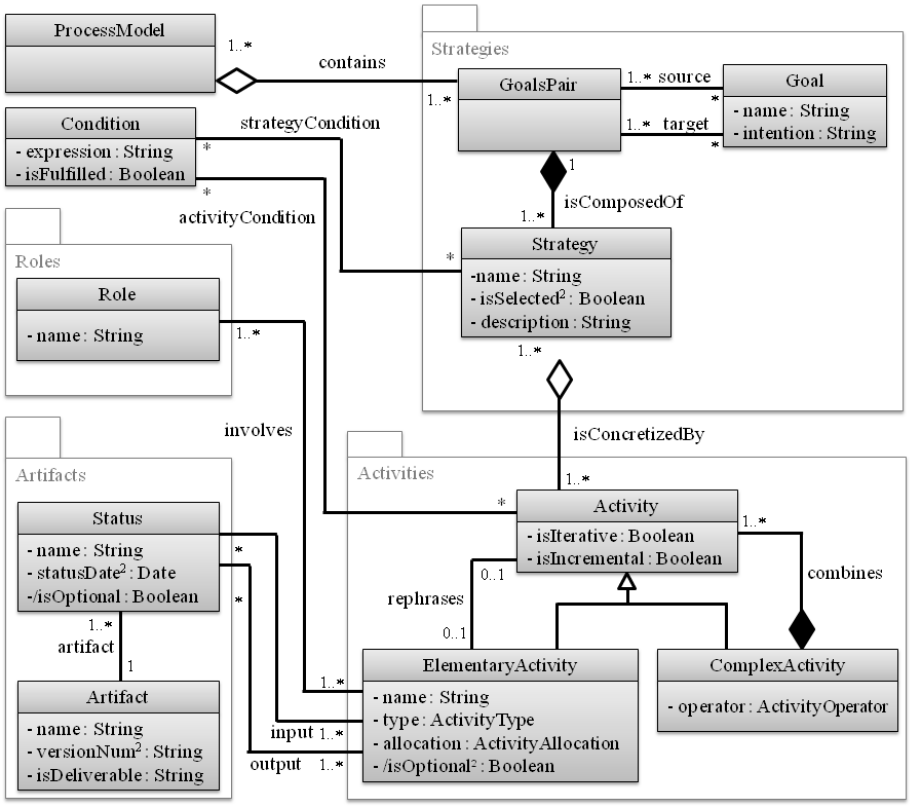

Figure 1. M2Flex metamodel overview.

[8] As we focus here on flexibility, all packages, classes and attributes are not shown. In all diagrams, the attributes whose name is followed by 2 (e.g. status2) are "simple fields" of "deep instantiation" (Atkinson and Kühne, 2001): when reifying the metamodel, they are instantiated into identical attributes at model level (and, as usual, into values at object level). We use this mechanism to impose, at the metamodel level, attributes that are needed at model level.

We want to express that various strategies can lead from one main stage of the process model to another. Inspired by the work done for defining the Map metamodel (Rolland et al., 1999), we model this package with goals and strategies. A process model is therefore considered to be composed of some main *GoalsPair*s, that represent couples of *Goal*s, one being the a source and the second being the target.

A goal represents an important objective of a process model. For instance, the Cameleon top-down approach (Calvary et al., 2001) defines four stages: (1) define the task and context models, (2) generate the Abstract UI, (3) generate the Concrete UI and (4) generate the Final UI. These stages would be represented as goals in M2Flex. A goal has a name and an intention, which is a description of its purpose.

As it is possible to achieve any of these goals in many ways, a *GoalPair* is composed of several equivalent *Strategies*. For instance, a requirement analysis stage, modeled here as the goal "describe the requirements", could be reached using a User Centered approach (Norman and Draper, 1986) or the Map approach (Rolland *et al.*, 1999). This gives rise to a first form of variability.

At enactment-time, a *Strategy* can be selected or not by the designers: this is modeled by the isSelected attribute with deep instantiation. A *Strategy* can be associated to some *Condition*s that represent the constraints that have to be fulfilled before the strategy starts. For instance, in requirements analysis, a User Centered strategy requires the agreement of the customer and the availability of some end users.

Strategies are concretized into activities, representing the operational tasks to be realized. In order to represent various amounts of details and various organization of tasks, activities can be elementary (*ElementaryActivities*) or composite (*ComplexActivities)*. For instance, the complex activity "design the UI" could be composed of some activities such as "mockup the UI", "evaluate the UI", "Improve the UI". This gives rises to refinement needed in granularability. An elementary activity has a name, a type (requirement analysis, coding,...) and an allocation (human, interactive or system task). The isOptional attribute is computed at process model enactment-time. It means that the activity can be **not** executed, i.e. that there is a path in the process model that does **not** include this activity. This is another form of variability. An elementary activity can be expressed in

various languages thanks to its *rephrases* relation to an activity, constituting the second form of the process model granularability.

The *ComplexActivity* class offers operators between activities and elementary activities. These operators are based on the operators used in task modeling (Nóbrega et al., 2006). After analyzing which operators are relevant here, we defined a set of 5 n-ary operators: **sequential enabling** which requires that activity A is completed before activity B can start, even if B does not depend on artifacts produced by A; **parallel**, that allows the activities to be realized in parallel; **choice**, which enables the designer to choose between some equivalent activities, i.e. activities that produce similar results or outputs. An activity that can be chosen or not is optional, this is a form of completeness; **interleaving** which enables activities to be realized in parallel by a unique agent/role, who can switch from an activity to another when he wants; **disabling**, that disables the targeted activities when the source activity is achieved.

As activities often require that previous tasks have been completed and have produced some results, they are associated twice to the *Artifacts* package, once as inputs and once as outputs. More concretely, they are associated to the Status class, that represents the various statuses of an artifact. This makes it possible for an activity to depend (or to produce) on an artifact with a specific status, for instance a validated version of an UI. An artifact can be optional, this is another form of completeness. When an artifact is optional, the activities that input it are necessarily optional too, because the inputs they require might be not available. Conversely, if an artifact is produced by an optional activity, it is optional too.

Activities are also related to *Roles*, in order to express that some competencies might be needed to complete the task.

As mentioned before, M2Flex supports all forms of variability, granularability and completeness. Distensibility can not be addressed by a metamodel, because it is an issue of enactment time and thereby a question of tools and validation of the process. This is why M2Flex has been completed by constraints. For instance, one constraint expresses that, if an activity requires an artifact as input, there must be another activity that produces this artifact with the needed status. These constraints make it possible to verify the process consistency and validity and thus to add or to avoid elements of the process at enactment time.

We have compared M2Flex to existing process metamodels, and shown that M2Flex is the lonely one supporting full flexibility. Please refer to (Céret *et al.*, 2013a) for more details.

2.3 Application of the UsiXML process model

The UsiXML method (Vanderdonckt, 2005) proposes an approach and a set of tools for the generation and the execution of plastic UIs. This approach relies on the successive transformations of a task model into an Abstract UI, Concrete UI and then into a Final UI, while integrating, amongst others, the models representing the manipulated concepts and the context.

UsiXML is already supported by a wide set of various and efficient tools, but this set offers only a very partial flexibility. For instance, several tools can be used to create models and to prototype UIs, like SketchiXML (Coyette and Vanderdonckt, 2005), VisiXML or GraphiXML (Michotte and Vanderdonckt, 2008). These tools make it possible to create more or less detailed and precise prototypes, according to the design or development stage. The prototype can be transformed into a Final UI for various runtime environments, either by transforming the underlying models directly in these tools either using some plug-in or specific tools. This palette of tools offers thus choices, a first form of variability. However, they all require that the designers create some of the UsiXML models (e.g. user, environment, domain) and master the rational of these models.

(Bouillon *et al.*, 2005) also propose some flexibility with the ResersiXML tool. Indeed, this tool makes it possible to generate an AUI and a CUI from an existing UI, saving part of the effort required to learn the models, and bringing ways to reuse existing systems. But this tool has a limited scope, being devoted to Web only. Thus, the flexibility it offers is not generalized to all existing systems.

Despite the rich tools palette sustaining UsiXML, the flexibility is partial, existing knowledge is poorly exploited and the reuse of existing elements is limited. We propose to increase this flexibility significantly, by improving the process model. To achieve this, we have modeled the UsiXML process according to M2Flex, thanks to D2Flex, our tool dedicated to process modeling. Then we have added, granularability and completeness. Figure 2 shows some part of the resulting process model.

Variability has been added thanks to various strategies. Figure 2a shows goals and strategies. Raising the goal "Generate CUI" (white point #2 one the figure) after having achieved the goal "Generate AUI" (#1) can be done thanks to strategy "code AUI2CUI" (#3, meaning creating the needed transformations) or using the strategy "pick up in repository" (#4, meaning that transformations can be found in the repository and then adapted if needed). Variability also relies on choices, as shown on figure 2b, that details the strategy "Model tasks, domain and environment" (#5). The designer has to choose how he wants to create the domain model (#6): either by creating it (#7) or by generating it (#8).

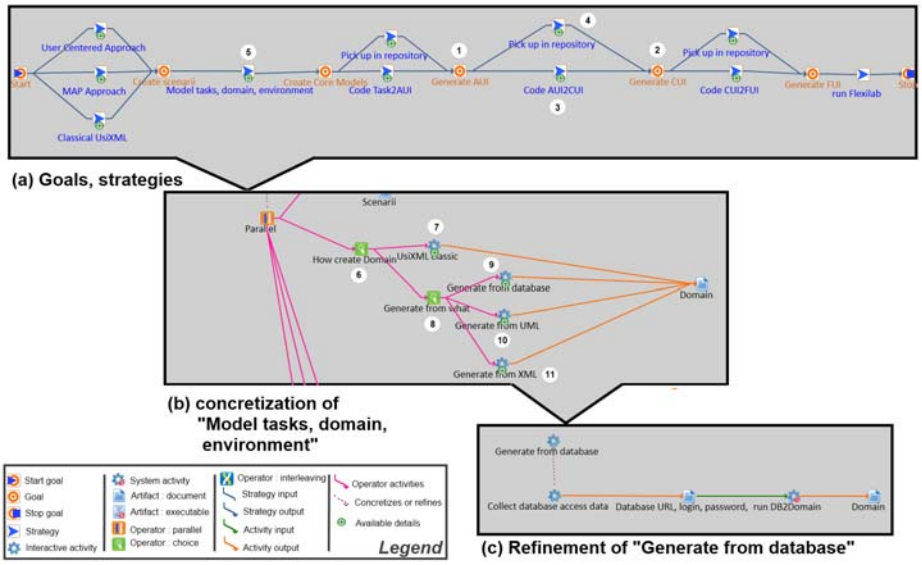

Figure 2. Flexibilized UsiXML process model drawn with D2Flex.

Granularability relies on refinements of activities: figure 2c shows a sequence of activities for achieving the generation of the domain model from a database.

Completeness relies here on the possibility to not create by hand all the models. For instance, if the designer does not want to create his domain model, he can generate it in several ways (#8), for instance by transforming the structure of a database into a class diagram (#9). Obviously, the resulting domain model might be of poor quality, depending on the quality of the database structure. However, it might be used as a first version of the domain model, and then improved if needed. Similarly, the designer could

choose to create the domain model by reusing some know-how he already masters, for instance by generating it from a UML diagram (#10) or an XML file (#11) which he knows how to make.

Several parts of the UsiXML process model has been "flexibilized", making it possible for designers and developers to enact activities requiring less expertise in Model-Driven techniques, and to reuse their competencies and know-how. This gives rise to decreasing the threshold of use of Model-Driven Engineering in UIs development and thus, to an increased quality of life.

3 Quality for the End User: Self-explanatory UIs

Many works (Lim *et al.*, 2009, Myers *et al.*, 2006, Purchase *et al.*, 2002) have reported on the benefits of supporting users through explanations in interactive systems. These explanations address specific questions that users ask about the User Interface (UI). For instance, how a task can be accomplished, why a feature is not enabled, or where an option is. Classical approaches (Horton, 1994), which are based on predefined information such as static documentation, FAQs, and guides, specify this information at design time. Their scope is therefore limited because users can have questions about the UI that are not covered by these kinds of supports. Moreover, this static documentation is not only a time consuming task but, additionally, it requires manual updates when the program specification changes. The problem is critical for plastic UIs where parts of the UI may be present or not at a given moment, or can be distributed on another device. To overcome this limitation, some researches (García Frey *et al.*, 2012) have recently proposed Model-Driven Engineering (MDE) as a means for supporting users at runtime. Model-Driven UIs use the models created at design time as their knowledge-base at runtime, exploiting the models and the relationships between them to find answers to the users' questions. These kinds of UIs with support facilities based on their own models are also known as Self-Explanatory UIs. Their main advantages are that answers are generated at runtime, and they evolve with the program specification automatically.

3.1 Model-Based Explanations

Model-Based explanations exists for different types of models an specific types of questions. An early example that employs a task model (in the form of user's actions) for explanation purposes is Cartoonist (Sukaviriya *et al.*, 1990). Cartoonist generates GUI animated tutorials to show a user how to accomplish a task, exploiting the model for providing run-time guidance.

(Pangoli and Paterno, 1995) allow users to ask questions such as *How can I perform this task?* or *What tasks can I perform now?* by exploiting a task model described in CTT. Contrary to Cartoonist, answers are provided in (Pangoli and Paterno, 1995) in natural language. Tasks modeled in the form of Petri Nets are used for similar purposes by (Palanque *et al.*, 1993), answering questions such as *What can I do now?* or *How can I make that action available again?*

Other works report on the usage of task models as a means for creating collaborative agents that help the user (Eisenstein J. *et al.*, 2002).

Behavioral models, presented in different forms, have been also used to support Why and Why not questions in user interfaces. In (Palanque *et al.*, 1993) Why questions are answered using the same approach based on Petri Nets that is exploited for procedural questions. By analyzing the net, it is possible to answer questions such as Why is this interaction not available?

The Crystal application framework proposed by (Myers *et al.*, 2006) uses a "Command Object model" that provides developers with an architecture and a set of interaction techniques for answering Why and Why not questions in UIs. Crystal improves users' understanding of the UI and help them in determining how to fix unwanted behavior.

Lim observed (Lim *et al.*, 2009a; Lim *et al.*, 2009b) that why and why not questions improve users' understanding and confidence of context-aware systems.

(Vermeulen et al. 2010) proposes a behavior model based on the Event-Condition-Action (ECA) paradigm, extending it with inverse actions (ECAA−1) for asking and answering why and why not questions in pervasive computing environments.

These researches show explanations based on individual models that propose different solutions for questions of specific types. We propose a set of design principles for homogenize and unify the way in which model-based explanations can be computed regardless the type of the question or

the underlying models of the user interface. These self-explanation principles are described in the next section.

3.2 Self-Explanation Design Principles

The infrastructure (figure 3) consists in two model-based UIs, the self-explanatory facility for providing the help and the UI of the underlying application. For a discussion on how to mix both sets of models see (García Frey *et al.*, 2012). The functional core of the help UI is composed of 4 modules for generating the list of questions (QG), interpreting (I) a user's request, i.e., inferring the type of question and its parameters, the processor (P) that computes the answer based on such parameters, and the answer generator (AG) that presents the answer back to the user. Each of these four modules of the functional core of the self-explanatory facility has full access to the models of the underlying application at runtime.

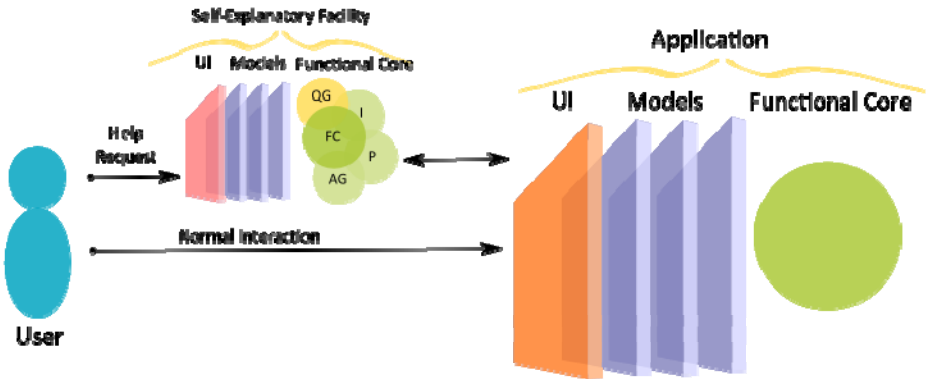

Figure 3. Infrastructure for self-explanatory UIs. The possible questions are generated by the Question Generator (QG) from the Functional Core of the help facility (FC). For each user's request, the Interpreter (I) determines its type and parameters, used by the Processor (P) to compute the answer, which is presented in some form (textual in this prototype) to the user thanks to the Answer Generator (AG). These four modules use the application models at runtime.

3.3 Explanation Strategies: Questions / Answers Computation

The infrastructure previously shown in figure 3 allows to compute different types of questions along with their associated answers at runtime. An Explanation Strategy describes this process for both questions and answers. We have built six different explanation strategies for types of questions. This section reviews some of them. For a more detailed information, please see (García Frey *et al.*, 2013). As an example of an explanation strategy, the next section describes how to compute *How?* questions along with their own answers, given a UI based on the UsiXML models.

3.4 Procedural Questions – How?

To generate How questions, we explore the task model recursively from the root task to the leaves. For each node representing a task, we create a question in a textual form according to the following grammar:

How to + Task.name + ?

where tasks are named starting with a verb following a standardized convention. An example of a How question is:

How to choose Packs?

Where "choose Packs" is the name of the task inside the task model of the UI of the application.

The computation of the answer is done as follows. First, we locate the task inside the task model. Second, we inspect the mapping model that maps tasks to AUI elements from the AUI model, so we can retrieve the abstract UI element that resulted from transforming such task. Once the AUI element has been found, we repeat the procedure to locate the CUI element derived from this AUI element. This is done by inspecting the mapping model that keeps track of the transformations from AUI elements to CUI elements. Once the CUI element has been retrieved, we compose the answer with following grammar:

Use the + CUI-elem.name + CUI-elem.type

An example of a computed answer using this approach is:

Use the Packs button

In this example, the CUI-elem.name is “Packs” and the CUI-elem.type is “button”.

Note that the answer can be completed with the information about the localization of the widget, which is computed also for Where questions. In this way, a more elaborated answer for CUI elements that were not directly visible from the user's were composed as follows:

Use the + CUI-element.name + CUI-element.type +
in the + CUI-element.parent + CUI-element.parent.type

where an example is:

Use the 'Pack Connected Drive' checkbox button in the 'Optional Equipment' panel.

4. Architecture

The software architecture of UsiComp relies on services. These services are implemented according to the OSGi specification. The main service is the Controller Service (figure 4).

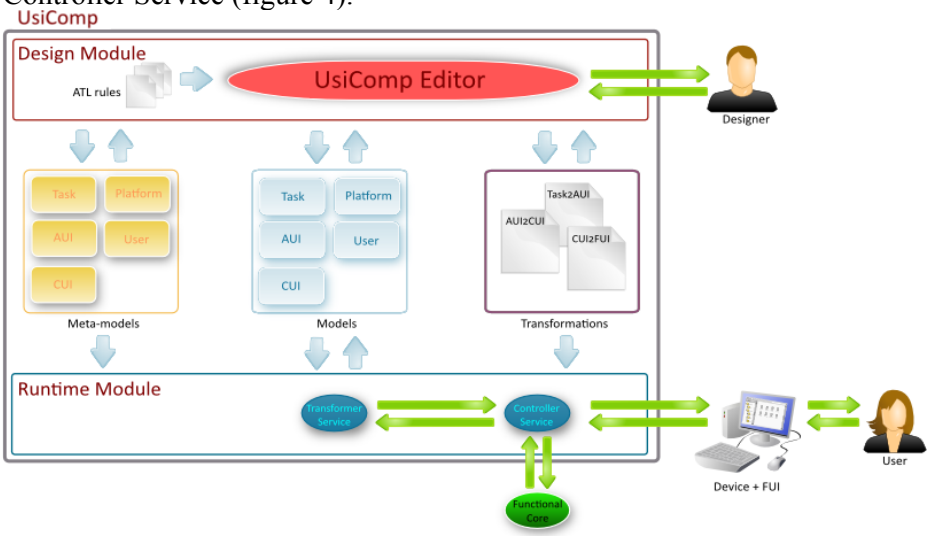

Figure 4. UsiComp software architecture: meta-models, models and transformations at the heart of both design time (IDE for designers) and runtime (FUIs for end-users).

The Controller Service is in charge of orchestrating the whole process in which a UI is generated by successive transformations. Transformations may be reifications or abstractions (Calvary et al., 2001). Reification (respectively Abstraction) lowers (respectively increases) the level of abstraction of a model. Currently, only reifications have been implemented and integrated into UsiComp. However, the architecture is fully generic, and so capable of integrating abstractions as well. UsiComp (figure 4) is made of two modules: one for design, another one for runtime. They share common resources: meta-models, models and transformations.

In order to support flexibility, UsiComp could not rely on a hard-coded generation process (e.g transform task into AUI, transform AUI into CUI and so on), that would have been unable to take into account the choices made by designer. For instance, in such a rigid generation process, the domain model could not be generated from a database. This is why we defined a model describing the stages to be executed for generating the UI. All steps are detailed and ordered accordingly to the decisions made by the designers.

4.1 Design module

The design module includes a visual editor for designing and prototyping UIs. The UsiComp editor offers the following functionalities:

- It allows designers to define all the models and transformations needed to produce a UI. Designers can create models by picking up the needed components and combining them.
- Transformations between models are composed of rules. A rule specifies how one specific set of elements of a source model is transformed into a set of target model elements. Designers can select what rules they want to apply to a given model, and the system will automatically compose the resulting transformation. Most common rules are already available in the system, but designers are free to add other rules if needed. Transformations and rules are written in the Atlas Transformation Language (ATL).
- The UsiComp editor verifies that the designed models comply with their corresponding meta-models. For instance, a binary operator in the task model must link two different tasks. The UsiComp editor

also composes and compiles the transformations and rules thanks to an integrated ATL compiler.

- The resulting Final UI, which is the code of the UI, can be directly executed from the IDE giving designers the opportunity to preview the generated UI.

4.2 Runtime module

The UsiComp runtime infrastructure is built on OSGi services. It works as follows:

- Once a new device becomes available to the framework (a specific client is installed into the device for this purpose), UsiComp identifies its specific platform model containing the platform details. The current version of UsiComp contains platform models specified by hand.
- The Transformer Service (Figure 4) is a generic transformation service that can apply any transformation to any model or models, producing models or text as output.
- To produce the UI, the Controller Service manages the transformations, their order of execution and their related models and meta-models, calling to the Transformer Service as many times as needed. The platform model is considered in the transformation process to produce an adapted UI.
- In the transformation process, the Controller weaves the functional core of the application into the UI, embedding the calls from and to the UI.

The models, meta-models and transformations involved in the generation are directly accessed by the Controller Service, which is also responsible of linking the application logic from the functional core to the UI and viceversa.

UsiComp has been entirely implemented in Java, EMF, and ATL. The development environment can be launched as a normal Desktop application or as a Web application embedded in an applet. Thanks to the OSGi services, it is possible to dynamically update the editor without stopping the

application. For instance, updating a service or replacing the transformation language for another one can be dynamically achieved.

5. Conclusions

This paper promotes flexibility of design and development process models, even at enactment time. The corner stone is M2Flex, a process metamodel that covers the four dimensions of flexibility: (1) variability, the ability of the metamodel to provide several equivalent choices, (2) granularability, the possibility of defining refined and/or rephrased components, (3) completeness, the possibility of defining optional components and pre-defined reusable results, and (4) distensibility, the capacity of the resulting process model to be extended or cut at enactment-time.

M2Flex is original by the flexibility it offers to designers and developers, not only at design time as it is classically done, but also at enactment-time which is new to our best knowledge.

We illustrated how this flexibility can improve the quality of life for designers, making it possible to reuse existing components (e.g. a database), know-how and knowledge (e.g. creating a XML file). Obviously, the UIs produced by such a flexible development process cannot be "perfect". However, thanks to the process flexibility, designers and developers can reuse parts of their know-how and competencies, and are able to transfer some existing components into the paradigm of models: it makes it possible for them to create a first, albeit imperfect, version of their UIs, that they can iteratively improve, acquiring step by step the needed competencies.

We also have shown how to dynamically support users' by generating questions and answers at runtime. We have shown how to use the underlying models of the UI through explanation strategies to compute such support. An example of explanation strategy has been provided to answer How? questions according to the UsiXML models and metamodels.

This way of automatically computing support at runtime allows to help users in plastic UIs regardless the context of use in which the interaction is taking place.

6. Perspectives

In the future, we plan to improve our tools (for instance, with validity checkers for the constraints to be satisfied). We also plan to create additional tools like for instance a module for executing the process models compliant with M2Flex. Attention will be paid to distensibility and to impact development tools by configuring and/or executing them. We also intend to make extensions sharable and reusable: for instance, if an activity is created by a team, it might be made available to others. Finally, as soon as this series of tools will be available, we plan to evaluate usage and to collect best practices.

We also intend to extend and complete the flexible design and development process, in order to integrate, in the one hand, activities for improving UIs while acquiring competencies and, in the other hand, activities for designers and developers who already have skills in MDE.

We will conduct evaluations, in order to more comprehensively estimate the reaction of designers when facing flexibility.

Our future work also includes to test how scalable model-based explanations are, either with a huge number of models or with a huge number of users requesting answers.

We will study how to support new types of questions and better support the current questions that we are able to compute.

We also plan to investigate the use of design rationale questions to support the learning of HCI design methods.

7. Acknowledgments

This work is funded by the european ITEA UsiXML project.

References

Atkinson, C., Kühne, T., 2001. The Essence of Multilevel Metamodeling, in: Gogolla, M., Kobryn, C. (Eds.), ≪UML≫ 2001 — The Unified Modeling Language. Modeling Languages, Concepts, and Tools, Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 19–33.

Barry, C., Lang, M., 2001. A Survey of Multimedia and Web Development Techniques and Methodology Usage. Ieee Multimed. 8, 52–60.

Basili, V.R., Rombach, H.D., 1987. Tailoring the software process to project goals and environments, in: Proceedings of the 9th International Conference on Software Engineering, ICSE '87. IEEE Computer Society Press, Los Alamitos, CA, USA, pp. 345–357.

Bendraou, R., Sadovykh, A., Gervais, M.-P., Blanc, X., 2007. Software Process Modeling and Execution: The UML4SPM to WS-BPEL Approach, in: EUROMICRO-SEAA. pp. 314–321.

Booch, G., 1993. Object-Oriented Analysis and Design with Applications, 2nd ed. Addison-Wesley Professional.

Bouillon, L., Limbourg, Q., Vanderdonckt, J., Michotte, B., 2005. Reverse Engineering of Web Pages based on Derivations and Transformations, in: Proc. of 3 Rd Latin American Web Congress LA-Web'2005 (Buenos Aires, October 31-November 2, 2005), IEEE Computer Society Press, Los Alamitos, 2005. pp. 3–13.

Calvary, G., Coutaz, J., Thevenin, D., 2001. A Unifying Reference Framework for the Development of Plastic User Interfaces, in: Little, M., Nigay, L. (Eds.), Engineering for Human-Computer Interaction, Lecture Notes in Computer Science. Springer Berlin / Heidelberg, pp. 173–192.

Céret, E., Dupuy-Chessa, S., Calvary, G., 2013a. M2Flex: a process metamodel for flexibility at runtime. Presented at the Research Challenges in Information Science (RCIS'2013), Paris, France, pp. 117–128.

Céret, E., Dupuy-Chessa, S., Calvary, G., Front, A., Rieu, D., 2013b. A taxonomy of design methods process models. Inf. Softw. Technol. Elsevier 55, 795–821.

Coyette, A., Vanderdonckt, J., 2005. A Sketching Tool for Designing Anyuser, Anyplatform, Anywhere User Interfaces, in: Costabile, M., Paternò, F. (Eds.), Human-Computer Interaction - INTERACT 2005, Lecture Notes in Computer Science. Springer Berlin / Heidelberg, pp. 550–564.

Eisenstein J., Rich, C. Agents and GUIs from Task Models. Information Science. (2002)

Fitzgerald, B., 1998. An empirical investigation into the adoption of systems development methodologies. Inf. Manage. 34, 317 – 328.

García Frey, A., Calvary, G. and Dupuy-Chessa, S. Users need your models! Exploiting Design Models for Explanations. In Proceedings of the 26th BCS HCI Group conference. Birmingham, UK. 12-14 September (2012) Garzotto, F., Perrone, V., 2007. Industrial Acceptability of Web Design Methods: an Empirical Study. J. Web Eng. 6, 73–96.

García Frey, A., Calvary, G., Dupuy-Chessa, S. and Mandran N. Model-Based Self-Explanatory UIs for free, but are they valuable? In Proceedings of the 14th IFIP TC13 Conference on Human-Computer Interaction (INTERACT'13), 2-6 September 2013, Cape Town, South Africa. 2013.

Hamid, B., Radermacher, A., Lanusse, A., Jouvray, C., Gérard, S., Terrier, F., 2008. Designing Fault-Tolerant Component Based Applications with a Model Driven Approach, in: SEUS. pp. 9–20.

Harmsen, 1997. Situational Method Engineering. University of Twente, Moret Ernst &

Young Management Consultants, Netherlands.

Harmsen, F., Brinkkemper, S., Oei, J.L.H., 1994. Situational method engineering for informational system project approaches, in: Proceedings of the IFIP WG8.1 Working Conference on Methods and Associated Tools for the Information Systems Life Cycle. Elsevier Science Inc., New York, NY, USA, pp. 169–194.

Horton, W. Designing and Writing On-line Documentation (2nd ed). New York: John Wiley & Sons. (1994)

Hug, C., Front, A., Rieu, D., 2008. A Process Engineering Method based on a Process Domain Model and Patterns, in: Proceedings, C.W. (Ed.), MoDISE-EUS. Montpellier, France, p. 126.

Lim B. Y., Dey A. K. and Avrahami D. Why and why not explanations improve the intelligibility of context-aware intelligent systems. In Proceedings of CHI'09, pp. 2119-2128. ACM. (2009a)

Lim B. Y. and Dey A. K. Assessing demand for intelligibility in context-aware applications. In Proceedings of Ubicomp'09, pp. 195-204. ACM. (2009b)

Michotte, B., Vanderdonckt, J., 2008. GrafiXML, a Multi-target User Interface Builder Based on UsiXML, in: ICAS. pp. 15–22.

Mohagheghi, P., Fernandez, M., Martell, J., Fritzsche, M., Gilani, W., 2009. MDE Adoption in Industry: Challenges and Success Criteria, in: Chaudron, M.V. (Ed.), Models in Software Engineering, Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 54–59.

Myers B. A., Weitzman D. A., Ko A. J. and Chau D. H. Answering why and why not questions in user interfaces. In Proceedings of CHI'06, pp. 397-406. ACM (2006)

Nóbrega, L., Nunes, N., Coelho, H., 2006. Mapping ConcurTaskTrees into UML 2.0, in: Gilroy, S., Harrison, M. (Eds.), Interactive Systems. Design, Specification, and Verification, Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 237–248.

Norman, D.A., Draper, S.W., 1986. User centered system design: new perspectives on human-computer interaction. Lawrence Erlbaum Associates.

Palanque P., Bastide R. and Dourte L. Contextual help for free with formal dialog design. In Fifth International Conference on Human-Computer Interaction. Elsevier Science Publisher. (1993)

Pangoli, S. and Paterno, F. Automatic generation of task-oriented help. In Proceedings of UIST'95, ACM, New York, NY, USA, pp. 181-187. (1995)

Potts, C., 1989. A generic model for representing design methods, in: Proceedings of the 11th International Conference on Software Engineering, ICSE '89. ACM, New York, NY, USA, pp. 217–226.

Purchase H.C. and Worrill J. An empirical study of on-line help design: features and principles. International Journal of Human Computer Studies. 56, 5. pp. 539-567 (2002)

Resnick, M., Myers, B., Nakakoji, K., Shneiderman, B., Pausch, R., Selker, T., Eisenberg, M., 2005. Design Principles for Tools to Support Creative Thinking. Work. Spons. Natl. Sci. Found. 25–35.

Rolland, C., Prakash, N., Benjamen, A., 1999. A Multi-Model View of Process Modelling. Requir. Eng. 4, 169–187.

Sukaviriya, P. and Foley, J. D. Coupling A UI framework with automatic generation of context-sensitive animated help. In Proceedings of UIST'90, ACM, New York, NY, USA. pp. 152-166. (1990)

Vanderdonckt, J., 2005. A MDA-Compliant Environment for Developing User Interfaces of Information Systems, in: Proc. of 17 Th Conf. on Advanced Information Systems Engineering CAiSE'05. Springer-Verlag, pp. 13-1