Model based self-explanatory user interfaces

Alfonso García Frey [1], **Sophie Dupuy-Chessa** [2], **Gaëlle Calvary** [2]

1. *Yotako S.A.*
   9, Avenue des Hauts-Fourneaux, 4362 Esch-sur-Alzette, Luxembourg
   alfonso@yotako.io

2. *Univ. Grenoble Alpes, CNRS, Grenoble INP, LIG, 38000 Grenoble France*
   Prenom.Nom@imag.fr

ABSTRACT. *User interfaces play an important role in Information Systems, particularly for their acceptance. But in Human Computer Interaction, perfect quality is an utopia. Despite all the design efforts, there are always situations the user interface is not suitable for: this claims for quality reparation. This paper explores self-explanatory user interfaces, i.e. user interfaces capable of "rephrasing" themselves so that to make them understandable by the user. The approach follows the principles of model-driven engineering. It consists of keeping design decisions contained in models alive at runtime so that to dynamically enrich the user interface by augmenting it with a set of possible questions and answers. Based on a problem space, this article details how to support self-explanation for free thanks to models. It also proposes a software infrastructure UsiExplain based on the UsiXML meta-models. An evaluation is conducted on a case study related to a car shopping website. It confirms that the approach is relevant especially for usage questions.*

RÉSUMÉ. *Les interfaces utilisateur jouent un rôle important pour les systèmes d'information, en particulier pour leur acceptation. Or en interaction homme-machine, obtenir une qualité parfaite est une utopie. Malgré tous les efforts qui peuvent être faits lors de la conception, il y a toujours des situations pour lesquelles l'interface utilisateur n'est pas adéquate. Cela nous force à envisager une réparation de la qualité. Dans ce cadre, cet article explore des interfaces auto-explicatives c'est-à-dire des interfaces capables de "se rephraser" pour devenir compréhensibles par les utilisateurs. Notre approche sur les principes de l'ingénierie dirigée par les modèles consiste à garder les décisions de conception contenues dans les modèles vivant à l'exécution pour enrichir dynamiquement l'interface utilisateur en l'augmentant par un ensemble de questions/réponses. Basé sur un espace problème, cet article détaille comment réaliser des auto-explication "pour rien" grâce aux modèles. Il propose aussi une infrastucture logicielle, UsiExplain, basée sur les méta-modèles de UsiXML. Une évaluation basée sur un cas d'étude d'un site d'achat de véhicules en ligne a été conduite. Elle confirme que l'approche est pertinente en particulier pour les questions d'usage.*

Ingénierie des systèmes d'information – n° 4/2017, 129-157

KEYWORDS: *user interfaces, model-driven engineering, models at runtime, self explanation*

MOTS-CLÉS : *interfaces utilisateurs, ingénierie dirigée par les modèles, modèles à l'exécution, auto-explication*

DOI:10.3166/ISI.22.4.129-157

1. Introduction

User interfaces play an important role in Information Systems (IS). They constitute the visible part of the technical IS. So they play an important role in its acceptance. But users often find problems while interacting with user interfaces (UIs). Questions about where an option is, how to accomplish a task, or why did something happen in the user interface naturally arise due to the imperfect quality of the UI. This problem of insufficient quality can be due to bad designs but, in general, this is not usually the case. Quality problems exist because as the user is not the designer, the user has a different understanding of the UI, so the user encounters different problems or obstacles during the interaction process. Moreover, even if the designer intends to achieve a good quality level in the UI, he/she cannot foresee all these problems and obstacles at design time because each single user has his/her own understanding of the UI. So it is impossible to provide support for all of the users at design time for all the situations they might be in. This problem can be seen as a gap between the intended versus perceived quality.

To limit this problem, one solution is to give explanations to users, particularly to explain design choices. Many works have reported on the benefits of supporting users through explanations in interactive systems (Lim, Dey, 2009; Myers *et al*., 2006; Purchase, Worrill, 2002). Different theoretical and practical works and tools try to provide users with support at runtime in order to overcome the gap (Lim, Dey, 2009; Vermeulen *et al*., 2010; Myers *et al*., 2006; Palanque *et al*., 1993), and thus, the lack of quality. However, even if a great amount of tools exist, there are still applications without support, or if the support exists, it is rather limited or specific to a predefined set of questions such as *Frequently Asked Questions*. In consequence, the **coverage** of the support is a problem in turn.

The lack of good help and support in most of today softwares is mainly due to a problem of **cost**. Software industry has become very competitive and one way to reduce costs is to speed up the design process and to simplify documentation at the minimum level (Delisle, Moulin, 2002). The existing solutions have an associated cost as they need to integrate the help facility itself or, as in most of the cases, develop the knowledge base that will be used to provide the explanations that support users in order to overcome the gap. This article explains how explanations can be generated from design models, making model-based help systems a good compromise with respect to the two requirements of coverage and cost. It proposes a new problem space for self-explanatory systems and describes our system, UsiExplain, and its evaluation which have been publihsed in (García Frey *et al*., 2012; 2013).

The article starts by describing related work in section 2. Then section 3 presents a car shopping website to serve as a support for illustration and evaluation. Afterwards, section 4 outlines our self-explanatory system which evaluation is reported in section 5. Finally we summarize our contribution and envision perspectives for future work.

2. Related work

This section describes work related to the support of users through explanations. It starts by giving an overview of existing solutions before categorising them thanks to a problem space.

2.1. Overview

During the development of computer science in the twentieth century, different computer science domains addressed the problem of supporting users in the interaction with the systems using some forms of explanations.

Question Answering (QA) systems have significantly contributed to the classification of different explanation types. In 1977 and 1978, Lehnert proposed (Lehnert, 1977; 1978) one of the most used question types classification for open-domain QA systems. Lehnert defined thirteen types of questions such as goal orientation, judmental or request. Not all of them are directly useful for help systems because these types were specifically designed for open-domain QA systems, but authors from expert systems as well as authors from closed-domain QA systems started to focus on some particular Lehnert's types.

According to (Jackson, 1998), an expert system is defined as "A computer system that emulates the decision-making ability of a human expert". Expert systems were firstly structured into two well distinguished parts: the inference engine and the knowledge base. The inference engine is fixed and independent from the expert system whilst the knowledge base is variable and is used by the inference engine to perform the reasoning. This division originated the sub-family of expert systems called *Knowledge-Base Systems* (KBS). KBS, also known as Rule-Based Systems, focus on the underlying information -or base of knowledge- represented or modelled inside the system itself. According to Gregor and Benbasat (Gregor, Benbasat, 1999), KBSs cover four different types of questions: *What is*, *Why* control or strategic questions where answers provide explanations about the systemś control behavior, and problem solving strategy, giving an insight into the design rationale of the system logic., *Why* questions that explain the processes taken by the system to come up with its results, and *Why* justification questions that provide "deep explanations" about design rationale. But the development of help systems relying on expert systems requires the implementation of the inference engine as well as the definition of the knowledge base for the specific application. So this implies an extra-cost.

To avoid this extra-cost problem, model-based explanation systems can provide a solution. They follow the same principle as the Expert systems: the knowledge base

that is used to provide such explanations are the design models of the Model-Based approach. So decisions taken at design time are used to provide explanations thanks to models.

An early example that employs a task model (in the form of user's actions) for explanation purposes is Cartoonist (Sukaviriya, Foley, 1990). Cartoonist generates Graphical UIs (GUI) animated tutorials to show a user how to accomplish a task, exploiting the model for providing run-time guidance.

Pangoli and Paternò (Pangoli, Paternó, 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, Paternó, 1995) in pseudo-natural language. Tasks modeled in the form of Petri Nets are used for similar purposes by Palanque et al. in (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, Rich, 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 are exploited for procedural questions. By analysing 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. (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 et al. (Lim, Dey, 2009; Lim *et al*., 2009) observed 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.

2.2. Problem space

The previous researches show explanations based on individual models. We aim to evaluate the suitability and added value of model-based approaches of UI, that can use one or more different models at the same time. In particular, we want to see whether these model-based approaches can generate more powerful explanations or have any extra added-value with regard to the previous isolated solutions. To better understand the differences between these model-based works, we propose the QAP (Questions, Answers, Properties) problem space (Figure 1). Each arrow represents an analysis axis of our problem space. For readability, colours and circles are added to categorize axes. For example, the "Intrinsic" axis concerns the presentation of questions.

Figure 1. The QAP problem space for Support Systems classification

The **Questions** section represents the input of the help system, i.e., how the user asks the system for information. Questions can be asked in many different ways, for instance, using some sort of natural language or tooltips among others. A question categorizes the form (presentation) and content (abstraction) of the user's request that the help system must address.

The **Answers** section represents the output of the help system. As with questions, answers can be provided also in different forms, such as text, images or animations. Similarly to the questions section, an answer categorizes the form (presentation) and content (abstraction) of the support provided by the system to the user.

The subdivision of each of the two first areas, questions and answers, into a Presentation and Abstraction sections, is motivated by the three classification methods of explanation types identified by Gregor and Benbasat (Gregor, Benbasat, 1999). In QAP, **Abstraction** represents the nature of the request that the system needs to deal with. It can address questions about the system usage (as for instance, How can I do this?, or Why this option is not enabled?) and the system Design rationale (for instance, Why these elements are ordered in this way? Why this message is red?).

Presentation means how questions or user's requests are integrated into the user interface. The presentation can be either intrinsic or extrinsic. In the intrinsic systems the question is added into the user interface and it uses some of its elements for the creation of the query or questions. In the extrinsic systems, the question is formulated via an external, non-integrated and independent interface, that does not necessarily

use elements of the user interface to specify the request. For instance, manuals and on-line help systems are examples of extrinsic ways of asking for information.

Abstraction area represents the type of knowledge involved in the support. The possible types are Representation, Structure, Task-Concepts, and Functionality.

Representation indicates that the type of support is related to the physical representation of the user interface. This kind of support is normally addressed to widgets or elementary interactors of the user interface.

Structure represents support about how the parts of the system or the user interface are arranged or organized. Questions related to navigation issues are classical examples of this axis (Where is...?).

Task-Concepts indicates answers about goals and their related concepts. This axis covers traditional support about goals such as What can I do now?

Functionality describes answers related to the functional core of the application. For instance, What did happen? for the questions area or "Is the result of the query" for the answers area.

The **Properties** section collects some features of help systems that are relevant in the context of this research. These properties are Extensibility, Dynamicity, and Initiative.

Extensibility means whether the support provided by the help system can be improved in some manner, for instance by adding annotations, or new sources of knowledge to the system.

Dynamicity indicates if the information provided by the help system to the user is generated at runtime, i.e., computed directly by using some source of knowledge. Answers or explanations that are not dynamically generated rely on predefined support that cannot be modified once the application is running, or in other words, that needs to be rewritten and updated by hand at design time.

Finally, the **Initiative** axis represents if the action of providing support is started by the system or on the contrary, it is the user who starts the supporting activity by requiring help.

We mapped different works on QAP problem space:

– The Myers' Crystal application framework (Myers *et al*., 2006) provides an architecture and interaction techniques that allow programmers to create applications that let the user ask a wide variety of questions about why things did and did not happen in the user interface, and how to use the related features of the application without using natural language. The "Why" and "Why not" questions supported by Crystal are related to user's actions. So Crystal supports questions about the Usage of the system (Figure 2, blue line, axis Question-Abstraction-Usage). On the contrary, this system does not support questions about the Design Rationale. The explanations provided by Crystal are embedded into the UI and can even use elements

of the UI for some answers (axis Answer-Presentation-Intrinsic). The same applies for how users ask questions (axis Question-Presentation-Intrinsic). Answers are also extrinsic because the message support can be shown in a separated window (Answer-Presentation-Extrinsic). Finally, Crystal uses a Command Object model to implement all the actions. The commands the user executes are stored on a command list which serves as a history of all the actions that have been taken, and used later for answering "Why" and "Why not" questions about user's actions (axis Answer-Abstration-Tasks-Concepts).

– Vermeulen's PervasiveCrystal system keeps the idea of the previous Crystal framework but adapts it to pervasive environments. The authors state that there is no pervasive computing frameworks available that supports why and why not questions about the behaviour of the system. Moreover, existing desktop implementations such as the previous Crystal system cannot be easily integrated into pervasive computing frameworks, since the assumptions underlying these implementations rarely hold in pervasive computing. For instance, pervasive environments usually rely on multiple machines from which events originate. As an example, consider the situation where the user starts playing a movie on the TV and the lights go out. The system involves at least, the TV, the lights of the room, the sensors, and the system processing such events.

– Sukaviriya's Cartoonist system automatically generates help for explaining how to accomplish tasks. The explanations given by Cartoonist are provided in the form of animations. Cartoonist employs a task model (in the form of user's actions) for generating such animations. The animations constructed by Cartoonist show how to invoke the commands of an application. The mouse pointer is explicitly represented by a graphic. This graphic moves around the user interface, picking the objects from a panel of elements, and setting them up to complete specific t asks. T he questions supported by Cartoonist are then related to the Usage of the user interface. As the answers supported by Cartoonist cover information about "How can I ..." questions, this systems relies on the Tasks-Concepts axis.

– Lim *et al.* propose the Intelligibility Toolkit for asking several questions about user interfaces in the context of ubiquitous computing. According to (Lim, Dey, 2010), "The Intelligibility Toolkit makes it easier for developers to provide many explanation types in their context-aware applications. This ease also allows developers to perform rapid prototyping of different explanation types to discern the best explanations to use and the best ways to use them." The Intelligibility Toolkit tries to make context-aware applications intelligible by "automatically providing explanations of application behavior". To this end, the toolkit provides automatic generation of eight explanation types (Inputs, Outputs, What, What If, Why, Why Not, How To, Certainty) for four different decision model types (rules, decision trees, naiive Bayes, hidden Markov models). All the Wh- explanation types along with the Outputs and Certainty types are related to the behaviour of the application. For instance, "What if" explanations "allow users to speculate what the application would do given a set of user-set input values", and the "Why" explanations inform users "why the application derived its output value from the current (or previous) input values".

Figure 2 shows the resulting overlapping of the precedent works. This overlapping gives a global overview of where most of the works have currently focused for supporting users. Three main areas of interest have been identified as they are uncovered by the reviewed literature.

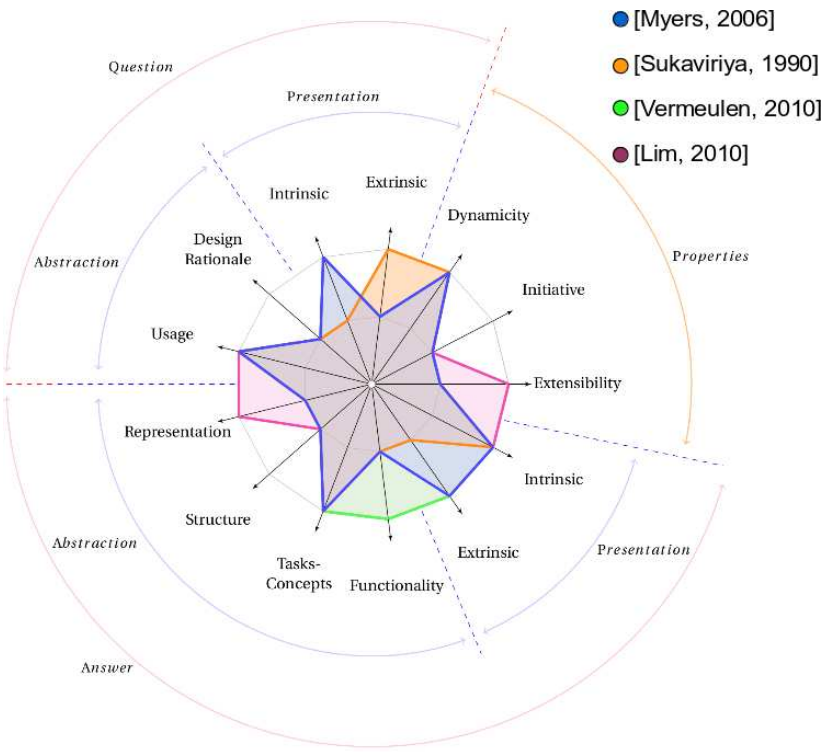

Figure 2. Overlapping the related work

In term of coverage, the first area of interest concerns the fact that most systems have a dedicated purpose. There is an implicit problem of unification. In fact, each of the reviewed model-based works such as Crystal or PervasiveCrystal addresses a specific type of questions, but we are not aware of any work that currently unifies different types of explanations at the same time. An approach for asking questions either for the usage or design rationale in a homogeneous way becomes necessary for supporting different question types simultaneously. In the same way, providing different types of answers require to uniform the way in which these answers are computed.

Secondly, we note a lack regarding the Design rationale of the user interface. The axis is not covered by any work. We did not identify any previous research able to provide questions about the design rationale of the user interface, that can help users to better understand "Why" the user interface is the way it is. Design rationale

questions can also be potentially useful for specific learning purposes in, for instance, a user interface design training course. To address this particular area, we will try in this research to enrich help systems, i.e., the types of questions that users can ask, with information related to the design decisions that are made at design time.

Finally, the third area of interest concerning coverage is the Structure. Information about the structure of the user interface is not usually provided. Structural information about the user interface can help to explain how the parts of the system or of the user interface are arranged or organized, so for instance, navigational questions could be finally supported in a model-based approach. In the context of this research we will explore how we can extract structural information from models to better support the user with this type of information.

Another uncovered axis is related to the Initiative. Initiative has widely been covered by agents as shown in the state of the art. For this reason, we will not directly address this issue from a model-based point of view but, instead, we will study what types of questions can take benefit from active help systems, to open other potential research questions.

Moreover some systems allow developers to choose the Presentation of Answers either in an Intrinsic or Extrinsic way. However, this is not the case for Presentation of the Questions, imposing one of both alternatives to the design of the user interface. In (García Frey *et al.*, 2012), we have explored how to overcome this limitation so designers and developers can fully customize how both questions and answers are integrated into the system.

So in the remainder of this paper, we will show how we address the coverage problem. The description of our work, will be based on a case study described in the next section.

3. Case study

Figure 3 illustrates a car shopping website where the user can select the type of vehicle (for instance, break, coupe or cabriolet) and configure the selected type of vehicle (engine options, colours, equipement...). After this process, the selected vehicle is shown according to the configured options (Figure 3, left side). One problem in this car shopping website is that the process does not always show the car as the result of the previous steps (Figure 3, right side). Moreover, in these situations there is no feedback to help users to understand what the problem is and/or how to fix it. As a consequence, users do not know if the selected configuration is not compatible with the car model, if they just missed one (or more) required configuration options, if some of the configuration options are not compatible with each other, or any other possible reason that avoids the selected and configured car to be displayed. In addition, some extra equipment is simply added by default with certain car models whereas with other car models the same options can be either available or not, and it is no longer the system that chooses these options but the user instead. For instance, selecting the

"Sport Design" version of the "Cabriolet" will add the "Bluetooth interface" for mobile phones but will suppress the "Sport Leather" option of the wheels. In this UI, the bluetooth option can be added regardless the version of the model car, but the second cannot.

Moreover, novice users could miss the meaning of some concepts that are used in the UI as for instance, what is the "Servotronic Direction at variable assistance".

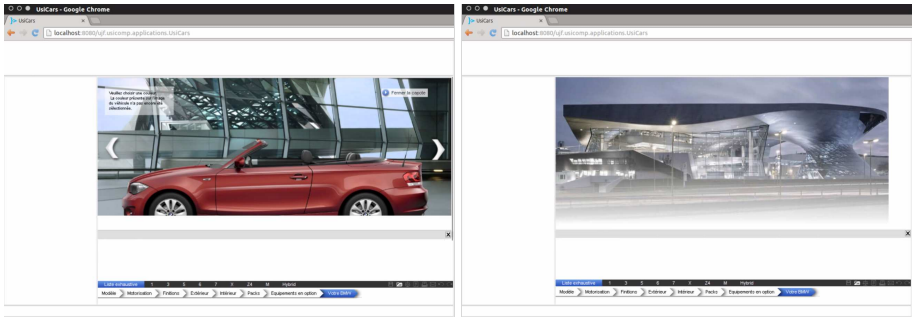

Figure 3. The car shopping website

4. Self-Explanations thanks to Model-Driven Engineering

This section presents the model driven approach we promote for producing self-explanatory user interfaces by design for free. It is illustrated on the case study.

4.1. Principles

Our approach to generate self-explanation consists of using the models done at design time to compute questions and answers at runtime that complement the UI (Figure 4). In its current version, it does not accomodate designer-defined question but it generates questions and answers from design models. The generated questions and answers constitute the self-explanation automatically generated at runtime. The self-explanatory facilities generated with our approach are responsible for:

– Generating the set of questions. We consider those questions that the help system "knows" how to answer by inspecting the underlying models of the UI. For this reason, it is convenient to generate these questions as well. By doing this, designers can propose to users the questions for which the system knows an answer.

– Generating answers. Once the user asks a question to the help system, the system needs to compute an understandable explanation or answer. This is done through the following three steps:

- Selecting the Explanation Strategy. In this phase the help system selects the explanation strategy that will be used for such a question (Figure 7). The explanation

strategy is selected according to the type of the question, meaning that the explanation strategy responsible for answering “How” questions, will probably inspect different models, retrieving different information, from the explanation strategy responsible for answering “Why” questions.

- Inspecting the models. Each explanation strategy will inspect one or more models to retrieve the elements that have been defined for each strategy. These elements will be used to compose the information that the user is requesting for.

- Composing the answer. Once all the elements of the models have been retrieved, the answer can be composed and prepared to be presented.

– Presenting the answer. The computed answer must be provided to the user in an understandable way. For the moment, we use a limited subset of natural language for questions and answers. We do not study their presentation, but we focus on their feasibility (are the question and answer computable?) and their functional utility.

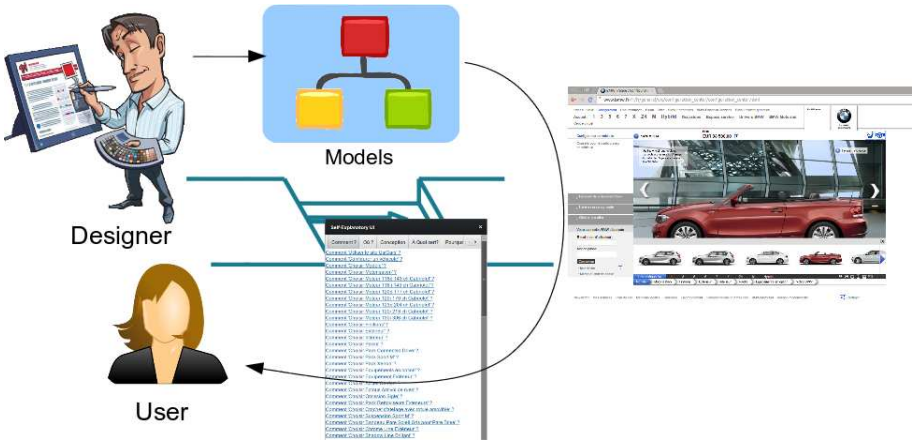

Figure 4. Principles of model-based explanations

4.2. Reference models

We use the models at all the levels of abstraction of the Cameleon reference framework. Cameleon (Calvary *et al.*, 2003) “characterizes the models, the methods, and the process involved for developing user interfaces for multiple contexts of use, or so-called multi-target user interfaces”. Cameleon is composed of four main levels of abstraction (Figure 5):

– Task and Concepts level that represents the tasks that the user can perform using the UI. The tasks manipulate domain concepts, also represented at this level.

– Abstraction level (with Abstract User Interfaces - AUI) that groups the tasks in a modality independent way, i.e., without taking into account the details of the platform or platforms where the user interface will be running after being generated.

– Concrete level (with Concrete User Interfaces - CUI) that defines how the user interface will be rendered. For instance, for Graphical User Interfaces, the Concrete UI defines what widgets are necessary for each element of the abstract user interface.

– Final UI level (with Final User Interfaces - FUI) that corresponds to the source code.

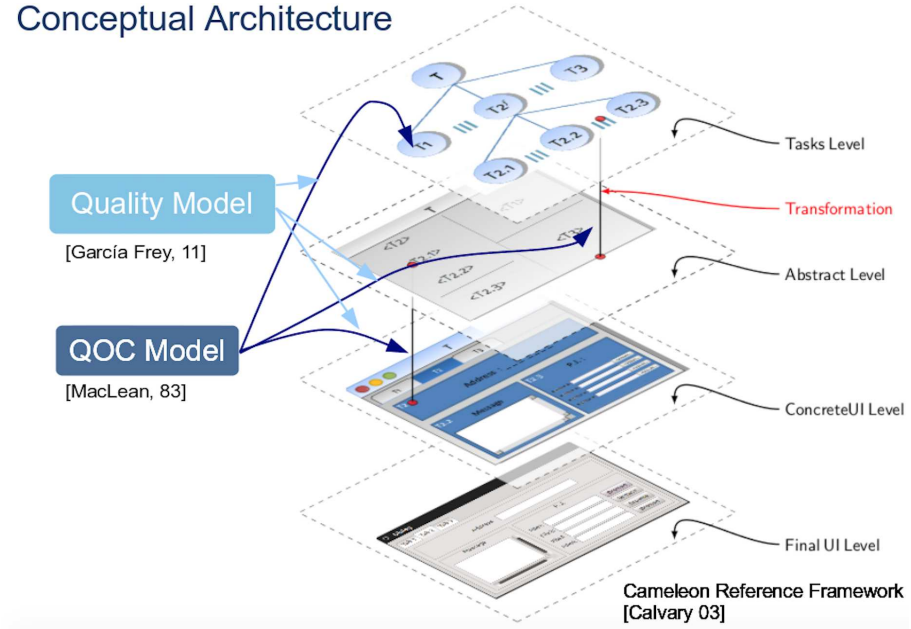

Figure 5. Models from the Cameleon Reference Framework and design rationale captured with QOC models

An example of Cameleon Reference Framework models for the self-explanatory system is given in Figure 6. A task model describes how the interactive system can answer a question. The identified tasks and their sequencing are transformed into an Abstract User Interface (AUI) model. Here the model is simple: for answering a question, only one interaction space is necesary; it contains a subspace for asking a question and another one for answering it. Then the AUI is in turn transformed into a Concrete User Interface (CUI) model representing the interactors or widgets. In the example, the interactors are a textfield for the question and a label for the answer. The Final User Interface (FUI) is derived from the CUI.

Inside the Cameleon's levels of abstraction, we use mainly models based on those from the UsiXML language[1], which preserves the four levels of abstraction, from the Tasks level (for the sake of simplicity, we use CTT in this paper at the task level) to the

1. For more information about the language, please visit http://www.usixml.org and http://www.usixml.eu

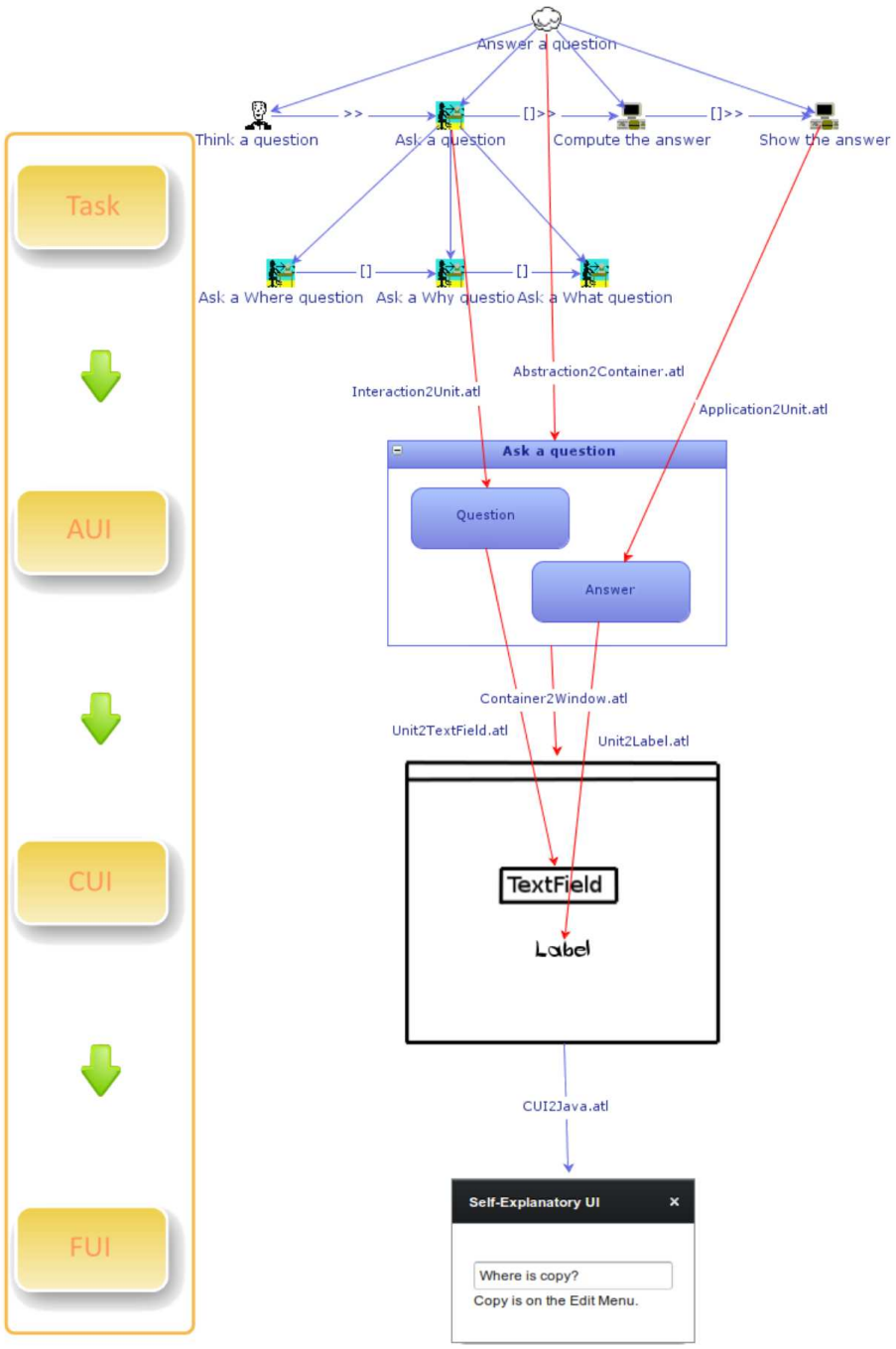

Figure 6. Models from the Cameleon Reference Framework for the car shopping website

FUI, reviewing their models and meta-models, and it extends the Cameleon Reference Framework with new models that add new functionality. In particular, we can note that the classical models of Cameleon are augmented with a QOC meta-model which supports design rationale based on the QOC (Questions, Options, Criteria) notation (MacLean *et al.*, 1991) and the Quality meta-model that we propose (García Frey *et al.*, 2011) to provide means for integrating quality criteria into the design process of the user interface. Indeed, design decisions are usually not arbitrarily taken. The QOC model allows to capture these design decisions inside the model and justify them through quality criteria if required. We also capture the way models are transformed along the four levels of abstraction modeling the Mappings between models.

Our help system is built from models. So it is independent from the domain, but its scope is determined by the models used. If models contain semantic information such as in QOC, questions can also be focused on semantic. In the same manner, if a model, for instance a CUI model, is related to an interaction style, this information can be used for generating questions. In its current version, only some models are used using the strategies presented in the next subsection.

4.3. Self-Explanation Strategies

Given a question, the self-explanatory user interface needs to retrieve the necessary information from the underlying models of the user interface, and compose the answer based on that information. This is the role of the *Explanation Strategies* presented in this section. An explanation strategy is responsible for computing the answer that corresponds to one specific type of question. In consequence, different types of questions are then managed by different explanation strategies.

4.3.1. Determining the Appropriate Explanation Strategy

We define different explanation strategies, one for each of the different types of questions supported by our self-explanatory help system. These questions are also generated by the self-explanatory facility. We currently support six different types of questions that have been reiteratively used by one or more approaches as shown in the state of the art. We built all these explanation strategies upon the main models of the Cameleon Reference Framework. The question types are summarized in Table 1.

Table 1. Question types

Question type	Question	Example
Procedural	How	How to select car packs"
Purpose or Functional	What is it for	What is the "Optional Equipment" button for?
Localization	Where .	Where is the tuner DAB?
Availability	What can I do now.	What can I do now?
Behavioural	Why/Why not	Why I cannot visualize the car?
DesignRationale	Design rationale of the UI	Why the engines are ordered by price?

Once the user asks a specific question, the self-explanatory facility will automatically retrieve the type of the question to determine which explanation strategy needs to be launched, and thus, what models will be inspected.

Figure 7 gives an overview of the different models that are involved in the generation of the questions with their respective answers. For instance, procedural questions such as "How to select a car?" are all generated using the task model, which is represented with the link between the Procedural box on the left side of the image, and Task Model in the centre. Answers to procedural questions are computed by using elements of the Mapping, Tasks, Abstract UI, and Concrete UI models, as represented in the image with the four links from the procedural box on the right side of the image to each model in the centre. In this way, our approach is not limited to web sites and can be used for any UI described in term of Mapping, Tasks, Abstract UI, and Concrete UI models.

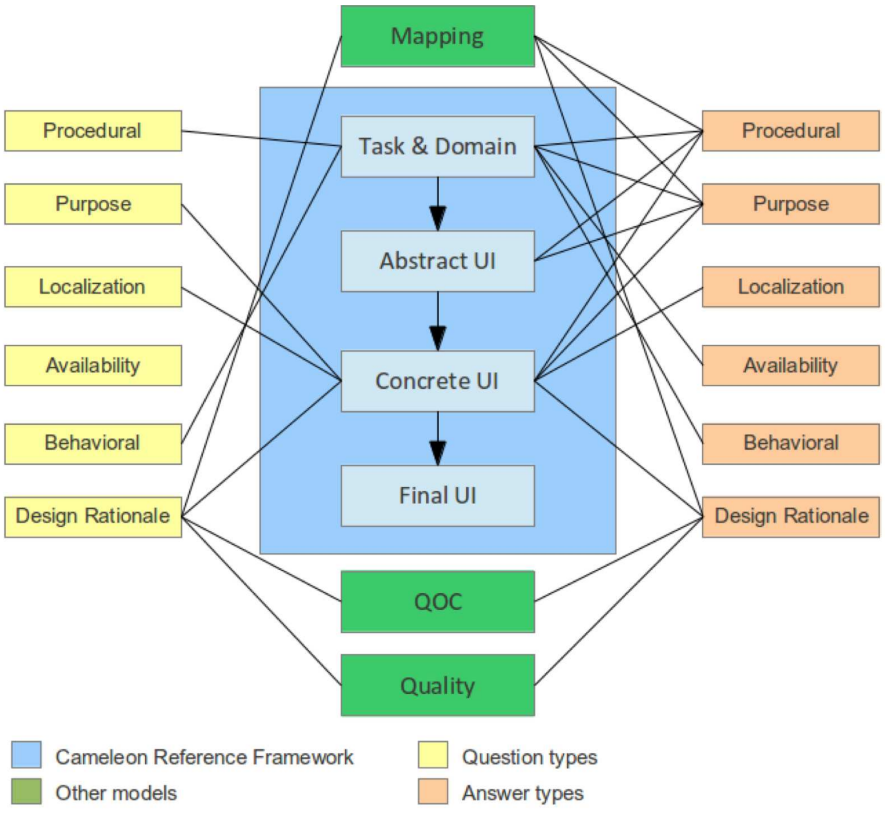

Figure 7. Models used for generating questions (left) and answers (right)

In the following, we detail the different explanation strategies that we have developed for each of the six question types. For each of them, we provide an explanation of how the questions of such type are generated, and how the answers are computed.

4.3.1.1. Procedural Questions - How

This section explains how to develop an explanation strategy to support How questions (Figure 8). How questions are requests that ask for the way in which a task can be accomplished. For instance, for the car shopping website a user can ask "How to select Packs?". The information that the user expects is the description of the procedure to accomplish the task, in the example, the instructions that show the user how to select different packs for a car.

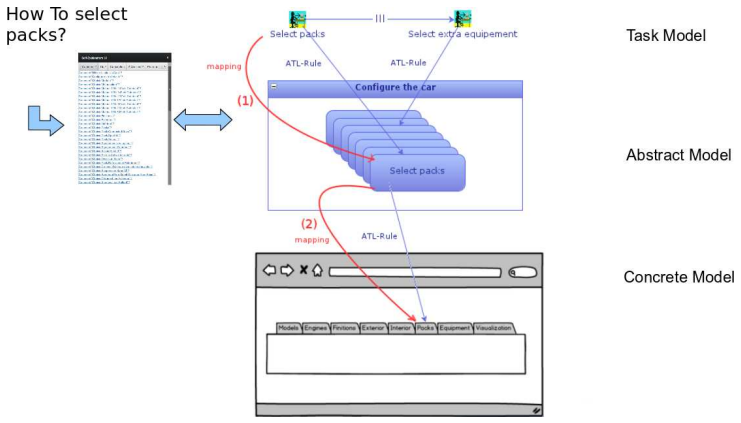

Figure 8. Explanation Strategy for answering How questions

We use the CTT notation in which there are four kinds of tasks: User tasks, System tasks, Abstract tasks and Interaction tasks. During the interaction with the system, users perform interactive tasks by using the elements of the UI. In other words, an interactive task is always mapped to one or more interactors at the CUI level during the transformation process. It makes then sense to generate questions of the form:

How to + Task.name + ?

where the task is an abstract or an interaction task.

To generate this type of questions, the explanation strategy can then explore the task model recursively from the root to the leaves. For each node representing an interaction task, the explanation strategy creates a question in a textual form according to the previous grammar. For instance, from a task tree, with a task "Configure the car" decomposed into two subtasks "Select Packs" and "Select Extra-Equipment", there are three generated questions: how to configure the car? how to select packs? how to select extra-equipments?

A possible way of answering a procedural question is then to indicate to the user what are the interactors that he/she needs to interact with in order to accomplish the

requested task. A possible way of answering a procedural question is then by retrieving through the models transformation the interactors in the CUI model, starting from the requested task at the task level. The composition of the answer is done according to the following grammar:

*Use the + CUI-element.name + CUI-element.type [+, CUI-element.name + CUI-element.type]**

By construction, there is always at least one CUI element an interaction task is transformed into. For example, a computed answer for the "How to select packs" question using this approach is:

Use the Packs tab

where the CUI-element.name is "Packs" and the CUI-element.type is "tab".

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

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

where an example is:

Use the Pack Connected Drive checkbox button in the Packs tab

Here, the CUI-element.parent.name is "Packs" and the CUI-element.parent.type is "tab".

4.3.1.2. Purpose/Functional Questions - What is it for

The purpose questions generated in the prototype were of the form:

'What is the + CUI-element.name + CUI-element.type + for?'

An example of a purpose question is:

What is the 'Optional Equipment' button for?

To compute these questions, we iterate through the CUI model of the UI, adding a question for each new element. We added questions for all the CUI elements except for layouts, as they are the only CUI elements that are not directly visible by the user. Answers where computed as follows. First we inspect the mapping model between the AUI and the CUI models to retrieve the AUI element from which the CUI element has been generated. Once we have the AUI element, we retrieve the task originating this AUI element, i.e., the source of the transformation chain. Once the task has been retrieved, we directly provide the name of the task, answer is computed using the name of the task in the following grammar:

To + task.name

As in the example:

To 'Select the optional equipment'

Even if this question is mostly useful for images or icons that have an unclear meaning, we also generated the questions and answers for the rest of the CUI elements, even if they presented textual information that made clear the purpose of the object.

4.3.1.3. Localization Questions - Where

The generated Where questions are of the form

'Where is the + CUI-element.name + ?'

As in the example:

'Where is the Tuner DAB?'

The process of generating these types of question is quite similar to the previous purpose questions. We only considered CUI elements having textual information, i.e., labels, any kind of textual buttons such as normal buttons, checkboxes or radio buttons, menus, menu options, and window titles. The reason for avoiding other types of widgets like images is that we did not want the user to describe such widgets and thus, asking open questions that the system could not understand. Answers were computed by finding the direct parent or container of the CUI element. This is, we first locate the CUI element in the CUI model and then we retrieve its parent, avoiding again layouts that are not visible for the user. For the Where question given in the previous example, the Tuner DAB refers to a checkbox button located on the 'Optional Equipments' panel. Thus, the grammar generating the answer is:

'The + CUI-element.name + is on the + CUI-element.parent + CUI-element.type'

So the answer given by the system is:

The Tuner DAB is on the Optional Equipment Panel

4.3.1.4. Availability Questions - What Can I Do Now

The "What can I do now?" question provides information about what tasks are currently available to the user regarding its current situation in the UI, i.e., depending on the current task that the user is currently performing at the moment of asking the question. As not all the tasks are always available at the time, answers for the same question can vary in time. The presented question is then always of the form:

What can I do now?

The computation of the answer relies on the task model. We first retrieve the current task in the task model. From the current task in the task tree, we compute which sister tasks are available regarding the LOTOS operators used by CTT. We add the name of each available task to answer. We then recursively iterate from the current task to the root task of the tree, adding all the available tasks. We finally add the

available sub-tasks. The final answer is then a list of tasks shown according to the next grammar:

You can + task-1.name + ... + task-N.name

For example, when the user accesses to the Optional Equipment panel, the answer to What can I do now? is:

You can select the external equipment, select the internal equipment, select the internal decorations, select the functional equipment, select the on-board electronics, select the wheel rims, select the maintenance contract.

4.3.1.5. Behavioural questions - Why I cannot

Behavioral questions were generated under the form:

Why I cannot + task-N.name + ?

Where the task task-N is unreachable from the current task. For instance:

Why I cannot Visualize the car?

To compute questions we proceed as for What can I do now?, locating the current task in the task tree first. We then locate all the unreachable tasks in a similar process, i.e., locating unreachable sister tasks (due to the CTT LOTOS operators) and traveling the task tree to the root and to the leaves. For instance, a task B enabled with information from a task A (A []» B) is unreachable until the information is received. Questions are added for all the unreachable tasks following the previous grammar. Answers are computed by finding the path that enables the given task. If a task is not reachable it means that some task or tasks need to be done. We find these tasks by traveling the sister and mother tasks (up to the root), locating the LOTOS operators that enable the desired task. For instance, for the task 'Visualize the car', the task was reachable by selecting the model of the vehicle first, so the provided answer is:

You need to Select the model

which conforms to the grammar we used:

*You need to + task-1.name [+, task-N.name]**

4.3.1.6. Design rationale questions

We included a QOC model to compute questions and answers about the design rationale of the UI. The proposed questions were directly retrieved from the QOC model. Answers were supported by one ergonomic criterion. The answers were computed according to the quality criteria supported by each option of the QOC model as shown in the following example:

Why the engines are ordered by price?

The provided answer, which follows the grammar “Because the ergonomic criterion is + criterion.description”, is:

Because the ergonomic criterion "Items of any select list must be displayed either in alphabetical order or in any meaningful order for the user in the context of the task".

Currently, these six types of questions (how, what is it for, where, what can I do now, why I cannot, design rationale) are implemented in our system for self-explanatory user interfaces, UsiExplain. Even if other types of questions can be added, the goal is to show the feasibility and the interest of the approach. Further researcher would be required to provide a complete taxonomy of question types.

4.4. Conceptual Architecture

The conceptual architecture (Figure 9) for self-explanatory UIs implements the explanation strategies described before and generates the self-explanatory UI. The generated UIs form a model-based help system -or self-explanatory facility- which completes the user interface of the target application. Both user interfaces are model-based, so they are both composed of the user interface models that are used to generate the user interface, plus the functional core, as depicted in Figure 9.

[t]

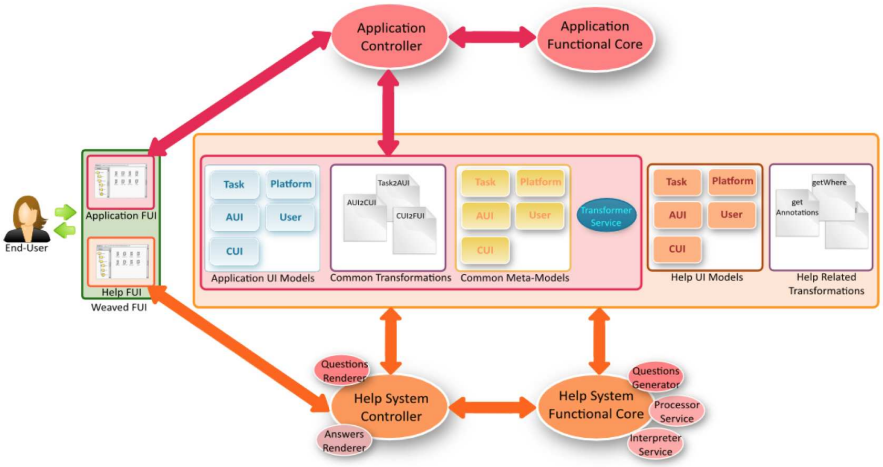

Figure 9. Generic architecture for model-based self-explanatory help systems

In this architecture, the UI of the application as well as the UI of the help system have their own Controller. The three vertical arrows in Figure 9 represent the access to the different model-related elements. For instance, the Controller of the application can access to the models, meta-models, and transformations of the user interface of

the application in order to generate the UI. In the same way, the Controller of the help system can also access the models, meta-models, and transformations of the UI of the help system for generation purposes. The functional core of the help system can access to any model-related element of both application UI and help UI, in order to find those elements that are necessary to compute the requested explanation.

From the end user's point of view there is only one UI. The question renderer is in charge of providing the user with a mechanism for asking questions. This could be completed by entering the question in natural language. For the moment, it is realized by selecting the desired one from a list of questions. When the user requests support, the help controller receives the request and passes it to the interpreter in charge of understanding the question. This interpreter could, for instance, parse the natural language input of the user or even recognise the gesture triggering the question with a gesture recognition system. The interpreter says to the processor what support information needs to be computed such as the type of the question and its parameters. The processor computes such information by accessing the models at runtime, according to the explanation strategy that has been specified for such type of question. This can be done by applying special help transformations that query the models at runtime, or by accessing the models via a special API as explained in the next section. The processor can query all the models independently if they belong to the application or the help system, and using exactly the same help transformations. This is possible because all the models conform to the same meta-models. Once the information has been retrieved from models and computed by the processor, it is prepared for the end user by the answers renderer. The answers renderer can update the UI with the desired information so the user can use it. The answers renderer is then responsible for managing how the information is presented. For instance, it could propose some text or voice using natural language, or an animation of the mouse cursor showing some procedure.

The architecture is prepared to extend the set of generic questions, according to the principles and the design of explanation strategies introduced in previous section.

4.5. Implementational Architecture

The conceptual architecture has been implemented in UsiExplain. UsiExplain is based on UsiComp (García Frey *et al.*, 2012), which supports models at design time and at runtime (Figure 10). UsiComp consists of a design module and a runtime module, both sharing common resources as meta-models and models. The design module includes a visual editor for designing and prototyping purposes. For the purpose of this paper we focus on the runtime side, that offers the following features:

– The Transformer Service is a generic transformation service that can apply any transformation to any model or models, producing models or text (code) as output.

– The Application Controller manages the transformations, their order of execution and their related models and meta-models, calling the Transformer Service as many times as needed. Its goal is to produce the UI.

– The Application Controller weaves the Functional Core of the application into the UI, embedding the calls from and to the UI.

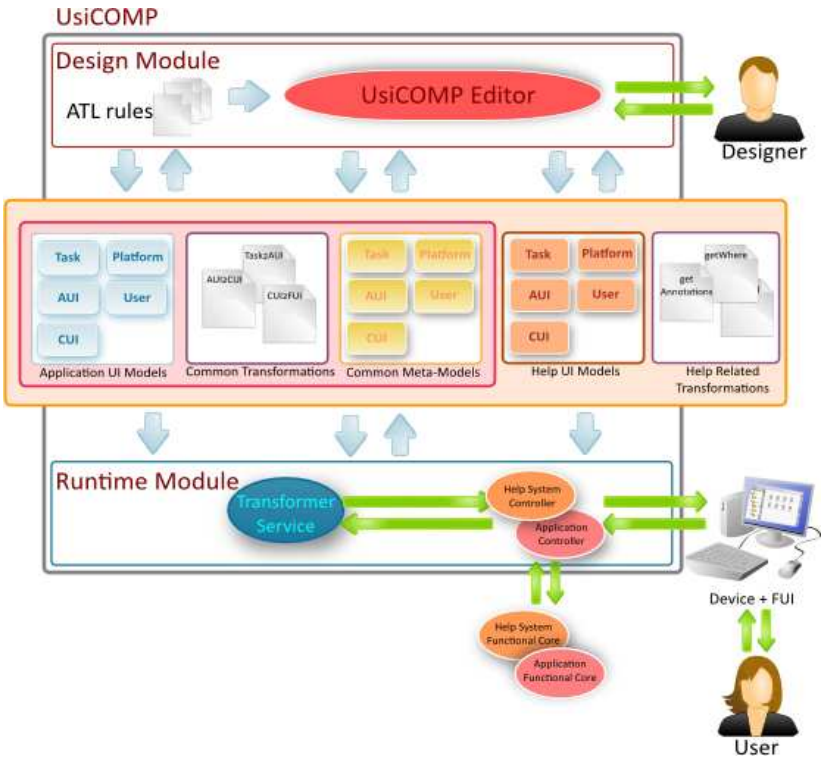

Figure 10. Implementation of UsiExplain with the UsiComp framework

UsiComp has been extended with features to generate a help system. A Help System Controller derives all the support requests from users to the Help System functional core, which applies the convenient explanation strategy and computes the answer accordingly, which is in turn resent back to the user.

The different elements of UsiExplain are implemented in Java, as OSGi services[2] to make it easier to incorpporate a multitude of different devices and ease the distributability of UsiExplain. Services include services for modeling with EMF[3], for transforming models with ATL transformation language[4]. The code of the resulting UIs, obtained by transformation, is pure Vaadin/Java framework.

2. http://www.osgi.org/
3. http://www.eclipse.org/modeling/emf/
4. http://www.eclipse.org/atl/

5. Qualitative evaluation

We conducted an experiment to evaluate the added value of model-based self-explanations. This evaluation has been published in (García Frey *et al.*, 2013). So we only summarize here its results to show that we have validated our working hypothesis.

5.1. Summary of the protocol

The experiment was conducted with a total of 20 participants, all between 23 and 39 with an average age of 27.4. From the 20 participants, 12 were male and 8 female. We recruited individuals regardless of their experience with interactive systems because the possible added value of model-based explanations can vary regarding the experience of each profile.

The experiment consists in using the improved car shopping web site with self-explanations. First we asked them to complete 10 different tasks in an established order. We asked the participants to "Think aloud": they verbalize their thoughts, specially the questions they would like to ask to the system and the problems that they find when accomplishing the tasks.

Then the prototype including a self-explanatory dialog that contained one type of question at a time is presented. The six questions (how, what is it for, where, what can I do now, why I cannot and design rationale questions) were presented one after another again in a randomized order. For each type of question, the dialog box showed all the possible questions that the participants could ask. Every time we showed a new type of question, we asked the participants their opinion about it, including the possible advantages and disadvantages of asking that question to the UI. We asked as well if the given type of question could be useful in the previous phase of the experiment. At the end of this phase, all the types of questions were shown together into the same self-explanatory dialog, and we asked some more general questions.

5.2. Findings

The last phase of the study revealed that questions of types How and Where were identified by most of the users (15/20) as useful and helpful with statements such as "It could be very helpful for locating all the options of the vehicle in a faster way". This last statement refers also to a gain of time, which was also identified as a positive value by a total of 10/20 users. The good acceptance of How questions contrasts however with the low number of verbatims. This suggests that users find the information useful but they are not thinking of asking it. The help UI could encourage/propose questions in these situations. The What is it for and Why questions were also identified as useful by an important number of participants, but less useful than the previous ones. This was mainly due to the fact that subjects did not find useful to ask for the purpose of some elements of the UI, such as check-boxes or labels, that already contain clear information about what they are currently doing. In the case of Why

questions, the results did not show a good acceptation by the participants as in the results found by (Lim, Dey, 2009; Myers *et al.*, 2006). This was due to the fact that the questions proposed by our algorithms did not cover all the possible range of questions that the participants asked. Finally, the What can I do now and design rationale related questions were found to be useless by most of the participants (16/20).

When we presented the help UI with all the types of questions together, the study revealed that in general, model-based self-explanatory facilities were identified as "useful" and "helpful" by most of the participants (16/20). The study also revealed question types that were not supported by our current implementation. The analysis of the collected data suggests that our model-based self-explanatory UI, with minor design enhancements for major usability improvements, could have the potential to easily help the users.

5.3. Work limitations

5.3.1. Unsupported types of questions

We identified other types of questions not explicitly supported by our s ystem. A minor number of them referred to What if questions. Even if most of these verbatims come from users that showed a trial and error approach to understand the consequences of their actions in the UI (i.e., they do not know the consequences of an action but they explore anyway to see what happens), 2 users out of 20 did not use options from the UI because they did not know their possible side-effects ("I have fear of loosing all the options"). Supporting What if questions can help this minority of users to feel more comfortable with the UI. These kinds of questions can probably be answered by analyzing the operators of the task model and how they are transformed to CUI elements, (what elements of the CUI model become active/inactive as we enable/disable new tasks.

We also identified a high number of verbatims requesting confirmation and validation from the UI. For instance, "does the car already have a navigation system?", "are the options included in the price?", were recurrent expressions used by the participants. This observation suggests that the feedback provided by the site was not enough for the users.

A third group of questions not supported by the self-explanatory dialog concerns definitions. Most of these questions were about specific car-related terminology and concepts such "What is the Tuner DAB?" or "What does Cabriolet stand for?". To support these questions, the proposed model-based approach needs to be extended with semantic information, either by adding new models or by connecting the UI with sources of semantic information (internet). Semantic information may be also necessary for answering questions about differences that we identified in a minor number of verbatims, for instance, What is the difference (or similarities) between the packs?
.

5.3.2. Usability Suggestions and Improvements

During the last phase of the experiment, where participants were confronted to the self-explanatory dialog, 14 out of 20 suggested that they would like to type the whole question directly instead of clicking on a predefined answer inside a list. 13 out of 20 would like to access questions by typing keywords in a text area, and 4 proposed to use a vocal interface instead. These observations sustain some of the design principles for help systems of the literature, in particular, “Help should be accurate, complete and consistent” (Shneiderman, Plaisant, 2010; Dix *et al*., 1998), and “Help should not display irrelevant information” (Horton, 1994). 6 participants suggested to classify questions not only by question types but following the categories of the underlying site, for instance, grouping them by equipment or car models. Regarding the answers, some participants argued that they do not like to read explanations, specially those that have a significant length. With the models used in this approach, the information given in the answers can be represented in non textual forms. Finally, some participants proposed that it would be preferable to use the questions not as a means to know how to find a specific option, but to "get there". This suggests that self-explanatory UIs could be used as software agents to overcome the usability issues of a UI not only by explaining to the user how to solve the issue, but solving it directly if possible. For instance, navigating to the desired website instead of explaining what website the user should navigate to. This observation opens new research questions: can self-explanatory UIs benefit from agents? If so, what other models are needed and how this can be done?

5.3.3. Scalability of the approach

We did not evaluate how the proposed solution performs in large scale applications with a high number of models. As the list of questions that the self-explanatory facility is able to answer, as well as the answers that it is able to provide, rely all of them on the underlying models using some parts of such models in the computation of the explanation strategies, a high number of models could have a significant impact in the performance of the help system. This potential problem is related not only to the self-explanation solution proposed in this thesis but to model-driven approaches in general. The scalability problem is not only related to the number of models but also to the com- plexity of them. For instance, the instance of CUI model containing thousands of objects could perform sensibly worse than an instance with less than a hundred objects.

6. Conclusion and perspectives

This paper studies to which extent models containing design decisions are suitable for supporting users in the interaction process by proposing explanations based on such models.The proposed approach is generic as new types of models can be considered in the explanation strategies without modifying our principles and the generic architecture supporting them. Moreover once explanation strategies are defined, they can be easily applied in different applications. But to be efficient, the approach re-

quires to design models precise enough to contain useful information. In its current version, our system could be improved in terms of usability and types of questions covered. It also need to be tested on larger systems.

If we consider the two requirements, coverage and cost, we can say that the goal is achieved. Considering coverage, we have shown how to include the design rationale and the structure questions, which had never been implemented so far (Figure 11). Moreover thanks to explanations strategies, all types of questions are unified and we can easily add new types of questions.

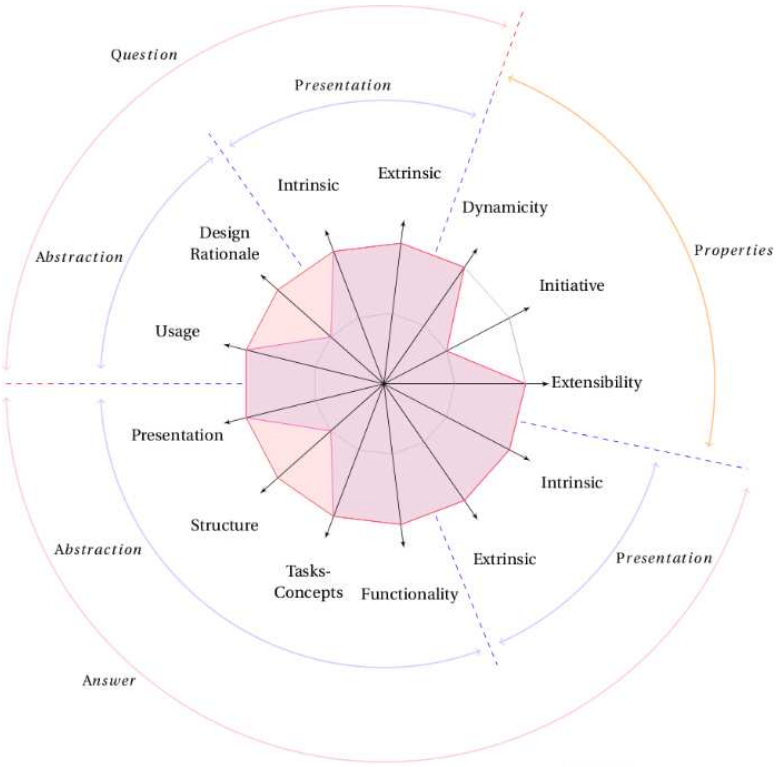

Figure 11. Comparison of our proposal to related work in QAP problem space

For the cost, help is free by construction as the system reuses design models. However this means that the models need to be complete or to be slightly adapted to the system to provide a good help.

Further research needs to be done to understand the specific efforts and burdens added for end users that could potentially hinder the adoption of the solution. We need to study the costs and the benefits for and users. We can explore what is the best presentation and integration of the proposed questions and their related answers. Improving the usability of the help system will lead to a better use of the help system and, in consequence, to a better experience with the target application. This research

will probably involve techniques for filtering questions out with regard to the user's profile, user's actions, or user's experience. The presentation of the answers should also be investigated, integrating techniques for exploiting answers in different ways. For instance, answers about localisation could take benefit of the model structure to directly propose the desired element to the user instead of providing the path that the user needs to follow to locate such element. The adaptation of the answer to each user should consider as well the use of different vocabulary if necessary, reviewing the quantity and nature of information provided to each particular user (more information for novice users, less for experts).

We also think about improving the quality of the UI by tracking what questions are asked by the users of a user interface. For instance, in the experiment presented in the previous chapter, almost 120 questions were related to navigational issues. This means that users did not find the option they were looking for easily. The user interface designers could study what question types are asked and at what precise moment, so they can later improve the user interface based on this information.

We can imagine to go a step further by letting the users' creating his/her own help system. From a model-based approach, the questions and answers presented by a self-explanatory UI could be considered as an extra-UI because they provide a different representation of the underlying models. End-user programming from a model-based perspective (e.g. as in (Dittmar *et al.*, 2012)) will help to explore other representations, eventually providing access to the full models of the UI if the user is an expert, or not only providing explanations about the UI but directly helping users to manipulate the models with an appropriate extra-UI with self-explanation support.

References

Calvary G., Coutaz J., Thevenin D., Limbourg Q., Bouillon L., Vanderdonckt J. 2003. A unifying reference framework for multi-target user interfaces. *Interacting With Computers Vol. 15/3*, pp. 289-308.

Delisle S., Moulin B. 200210. User interfaces and help systems: from helplessness to intelligent assistance. *Artificial Intelligence*, Vol. 18, No. 2, pp. 117–157. http://dx.doi.org/10.1023/A:1015179704819

Dittmar A., García Frey A., Dupuy-Chessa S. 2012. What can model-based UI design offer to end-user software engineering? In Proceedings of the 4th ACM SIGCHI symposium on Engineering interactive computing systems, pp. 189–194. New York, NY, USA, ACM. http://doi.acm.org/10.1145/2305484.2305515

Dix A., Finley J., Abowd G., Beale R. 1998. Human-computer interaction (2nd ed.)Human-computer interaction (2nd ed.). Upper Saddle River, NJ, USA, Prentice-Hall, Inc.

Eisenstein J., Rich C. 2002. Agents and GUIs from task modelsAgents and guis from task models. In Proceedings of the 7th international conference on Intelligent user interfaces, pp. 47–54. New York, NY, USA, ACM. http://doi.acm.org/10.1145/502716.502727

García Frey A., Calvary G., Dupuy-Chessa S. 2012. Users need your models! Exploiting Design Models for Explanations. In Proceedings of HCI 2012, Human Computer Interaction, People and Computers XXVI, The 26th BCS HCI Group conference (Birmingham, UK)..

García Frey A., Calvary G., Dupuy-Chessa S., Mandran N. 2013. 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. Springer.

García Frey A., Céret E., Dupuy-Chessa S., Calvary G. 2011. QUIMERA: a Quality Metamodel to Improve Design Rationale. In Proceedings of the third ACM SIGCHI Symposium on Engineering Interactive Computing Systems (EICS 2011), p. 265-270. ACM Press. http://dl.acm.org/citation.cfm?id=1996534

García Frey A., Céret E., Dupuy-Chessa S., Calvary G., Gabillon Y. 2012. UsiComp: an extensible model-driven composerUsicomp: an extensible model-driven composer. In Proceedings of the 4th ACM SIGCHI symposium on Engineering interactive computing systems, pp. 263–268. New York, NY, USA, ACM. http://doi.acm.org/10.1145/2305484.2305528

Gregor S., Benbasat I. 199912. Explanations from intelligent systems: theoretical foundations and implications for practice. *MIS Q.*, Vol. 23, No. 4, pp. 497–530. http://dx.doi.org/10.2307/249487

Horton W. 1994. Designing and writing online documentation: hypermedia for self-supporting products. Wiley. http://books.google.fr/books?id=qc9o_kV40NsC

Jackson P. 1998. Introduction to Expert SystemsIntroduction to expert systems (3rd ed.). Boston, MA, USA, Addison-Wesley Longman Publishing Co., Inc.

Lehnert W. 1977. The process of question answering. Yale. http://books.google.fr/books?id=J3e4AAAAIAAJ

Lehnert W. 1978. The Process of Question Answering: A Computer Simulation of Cognition. Erlbaum. http://books.google.fr/books?id=iupQAAAAMAAJ

Lim BY., Dey AK. 2009. Assessing demand for intelligibility in context-aware applications. In Proceedings of the 11th international conference on Ubiquitous computing, pp. 195–204. New York, NY, USA, ACM. http://doi.acm.org/10.1145/1620545.1620576

Lim BY., Dey AK. 2010. Toolkit to support intelligibility in context-aware applications. In Proceedings of the 12th ACM international conference on Ubiquitous computing, pp. 13–22. New York, NY, USA, ACM. http://doi.acm.org/10.1145/1864349.1864353

Lim BY., Dey AK., Avrahami D. 2009. Why and why not explanations improve the intelligibility of context-aware intelligent systems. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, pp. 2119–2128. New York, NY, USA, ACM. http://doi.acm.org/10.1145/1518701.1519023

MacLean A., Young RM., Bellotti VME., Moran TP. 199109. Questions, options, and criteria: elements of design space analysis. *Hum.-Comput. Interact.*, Vol. 6, No. 3, pp. 201–250. http://dx.doi.org/10.1207/s15327051hci0603\&4_2

Myers BA., Weitzman DA., Ko AJ., Chau DH. 2006. Answering why and why not questions in user interfaces. In CHI '06: Proceedings of the SIGCHI conference on Human Factors in computing systems, pp. 397–406. New York, NY, USA, ACM.

Palanque P., Bastide R., Dourte L. 1993. Contextual Help for Free With Formal Dialogue Design.

Pangoli S., Paternó F. 1995. Automatic generation of task-oriented helpAutomatic generation of task-oriented help. In Proceedings of the 8th annual ACM symposium on User interface and software technology, pp. 181–187. New York, NY, USA, ACM. http://doi.acm.org/10.1145/215585.215971

Purchase HC., Worrill J. 200204. An empirical study of on-line help design: features and principles. *Int. J. Hum.-Comput. Stud.*, Vol. 56, No. 5, pp. 539–567. http://dx.doi.org/10.1006/ijhc.1009

Shneiderman B., Plaisant C. 2010. Designing the User Interface: Strategies for Effective Human-Computer Interaction. Addison-Wesley. http://books.google.fr/books?id=2CfROgAACAAJ

Sukaviriya P., Foley JD. 1990. Coupling a UI framework with automatic generation of context-sensitive animated help. In Proceedings of the 3rd annual ACM SIGGRAPH symposium on User interface software and technology, pp. 152–166. New York, NY, USA, ACM. http://doi.acm.org/10.1145/97924.97942

Vermeulen J., Vanderhulst G., Luyten K., Coninx K. 2010. PervasiveCrystal: Asking and Answering Why and Why Not Questions about Pervasive Computing Applications. In Proceedings of the 2010 Sixth International Conference on Intelligent Environments, pp. 271–276. Washington, DC, USA, IEEE Computer Society. http://dx.doi.org/10.1109/IE.2010.56