HAL open science

Model-Based Self-Explanatory UIs for free, but are they valuable?

Alfonso García Frey, Gaëlle Calvary, Sophie Dupuy-Chessa, Nadine Mandran

To cite this version:

Alfonso García Frey, Gaëlle Calvary, Sophie Dupuy-Chessa, Nadine Mandran. Model-Based Self-Explanatory UIs for free, but are they valuable?. 14th International Conference on Human-Computer Interaction (INTERACT), Sep 2013, Cape Town, South Africa. pp.144-161, 10.1007/978-3-642-40477-1_9 . hal-00953351

HAL Id: hal-00953351

https://inria.hal.science/hal-00953351v1

Submitted on 28 Feb 2014

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution 4.0 International License

Model-Based Self-Explanatory UIs for free, but are they valuable?

Alfonso García Frey[1], Gaëlle Calvary[1], Sophie Dupuy-Chessa[2], Nadine Mandran[3]

[1]Grenoble INP , [2]UPMF, [3]CNRS, LIG
41 rue des mathématiques,
38400 Saint Martin d'Hères, France

{Alfonso.Garcia-Frey, Gaelle.Calvary, Sophie.Dupuy, Nadine.Mandran}@imag.fr

Abstract. Model-Driven Engineering (MDE) has been extensively used for generating User Interfaces (UIs) from models. As long as these models are kept alive at runtime, the UIs are capable of adapting to variations of the context of use. This paper investigates a potentially powerful side effect: the possibility of enriching the UIs with explanations directly generated from these models. This paper first describes a software infrastructure that supports this generation of explanations. It then reports on a user study that evaluates the added value of such model based self-explanations.
Keywords: Self-Explanatory User Interfaces, Model-Driven Engineering, Models at runtime, Self-Explanation.

1 Introduction

Many works ([9, 10, 14]) 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 [7], 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. To overcome this limitation, some researches [4] 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.

This paper firstly describes a self-explanatory system, explaining how to compute different types of explanations based on the underlying models. This system is representative of what explanations a model-based UI can provide. The paper then reports on an evaluation about the suitability of model-driven engineering for explanation purposes. We conducted a user study on such a self-explanatory UI. This UI reproduces an existent website, where all the necessary models have been generated by a reverse engineering process. The resultant model-driven UI has been extended with support facilities based on such models, to allow users to ask questions about the UI at runtime.

The paper is structured as follows. The next section provides related work on model-based explanations. Then, the paper describes the working hypothesis. Later, it presents the infrastructure of the self-explanation system that we used in our experiment. We also explain the algorithms that we used to, based on the models available at runtime, generate the questions and answers according to the model-driven approach. The paper then covers the user study, describing the experiment and the evaluation protocol. The final section analyzes and discusses the results of the qualitative analysis we did on the collected data. Finally, the paper ends with a conclusion and a brief discussion about the future perspectives.

2 Related Work

This section briefly explains the role of MDE in HCI and how its different types of models are used in different works for supporting users with specific types of explanations. A different type of explanation is provided regarding the nature of the question asked by the user. For instance, [15] describe five categories of questions: goal-oriented (*What can I do with this program?*), descriptive (*What is this? What does it do?*), procedural (*How do I do this*?), interpretive (*Why did this happen*?) and navigational (*Where am I? Where is it?*).

Other authors describe similar categories but with different terms. For instance in [17] we find conceptual explanations (*What is this?, What is the meaning of this?),* Why-explanations describing causes and justifications for facts, How-explanations for describing processes, Purpose-explanations (*What is this for?* or *What is the purpose of this?*), and cognitive explanations, which "explain or predict the behavior of 'intelligent systems' on the basis of known goals, beliefs, constraints, and rationality assumptions" [17].

Some of these types of explanations have been provided through different models in model-based approaches. We firstly explain the model-driven approach of UIs before presenting such model-based explanations.

2.1 MDE for HCI

Model-Driven Engineering (MDE) has been recently applied to the engineering of UIs. It consists in describing different features of UIs (for instance, tasks, domain, context of use) in models from which a UI is produced according to a forward engi-

neering process [19]. An example of a MDE-compliant approach is the Cameleon Reference Framework [2], where a task model is transformed into an Abstract User Interface (AUI) model, which is in turn transformed into a Concrete User Interface (CUI) model representing the interactors or widgets, from which the Final User Interface (FUI), i.e., the code of the UI itself, is derived. The experimentation presented later is based on this approach.

Some models of the Cameleon Reference Framework existed before MDE and they have been already used separately for explanation purposes in several works. For instance, task models have been extensively used for automatically generate procedural information in different forms.

2.2 Model-based explanations

An early example that employs a task model (in the form of user's actions) for explanation purposes is Cartoonist [18]. 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 [13] 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 [13] in natural language. Tasks modeled in the form of Petri Nets are used for similar purposes by Palanque et al. in [12], 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 [3].

Behavioral models, presented in different forms, have been also used to support *Why* and *Why not* questions in user interfaces. In [12] *Why* questions are answered using the same approach based on Petri Nets that is 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. [10] 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. [8, 9] observed that *why* and *why not* questions improve users' understanding and confidence of context-aware systems.

Vermeulen et al. [20] propose 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. 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. This is what our working hypothesis describes in the next section.

3 Working Hypothesis

In our previous work, we tried to combine different models for explanation purposes. We explored [5] how to let designers to describe the task model behind a UI, which is provided in the form of a Final UI (the model representing the UI itself), so procedural explanations could be provided based on this task model. We also explored how to support designers through design questions based on design rationale notations [6], and an architecture for the unification of several help facilities in [4], showing how to employ CUI models for *Where* questions, or annotated task models for providing descriptions at runtime.

These last works, that combine and integrate different models to compute several types of explanations, suggest that the model-driven approach of UIs, which is based on similar design models to those presented in the related works, should provide users with explanations that are at least of the same interest, usefulness and relevance than those provided by each model individually. Moreover, as model-based UIs can rely on a greater number of models or even include new ones, the power of explanation could be potentially better as well.

To verify the added-value of the model-driven approach of UIs for explanation purposes, we conducted an experimentation based on a model-driven prototype, which is described in the next section. The prototype was built according to the Cameleon Reference Framework, and it includes a self-explanatory facility in the form of a dialog to let users ask different types of questions about the UI. This system is representative of what explanations a model-based UI can provide.

4 Prototype Description and Supporting Infrastructure

This section firstly describes the prototype and the infrastructure of the help facility. Then it details the algorithms used for computing the questions and answers provided to the users, detailing how the models are used at runtime.

4.1 Prototype Description

The prototype consists in a cars shopping website called UsiCars. This website is inspired by a real site from a real car manufacturer. We reproduced only the part of the website that is devoted to the selection and configuration of the vehicles, keeping the options and the structure of the original website.

This website was chosen for two main reasons. The first one is that we needed to use a system that contains knowledge that is understandable and accessible by all the participants, but complex enough for not being easy to use. A website for configuring cars covered this point as all the participants understand many of the car related con-

cepts, but at the same time there are enough specific options with domain related concepts to create complex tasks that are non trivial to perform. The second reason is that we found the original website difficult to use by real users in different forums.

The reproduction of the website was done by a reverse engineering process. The first step was to explore all the different tasks that the user can perform to select and configure a vehicle. We created a task model according to this information. Secondly, we created a transformation to obtain an Abstract UI model that conforms to the structure of the original website. Thirdly, we wrote another transformation to generate the Concrete UI model from the Abstract UI model. This transformation produces all the widgets that we find in the original website. We also used the same images and we respected the same sizes for all the widgets from the original site, to ensure that we obtain the same usability properties. Finally, we wrote another transformation to generate the Java code and produce the resulting site.

In each one of the model to model transformations, we generated not only the target model but also mapping models that keep track of the successive transformations of an element from one model to another. For instance, in the transformation from the task model to the Abstract UI model we generated a *Mapping-Task2AUI* model that specifies what tasks are transformed into what Abstract UI elements. The same principle was applied to obtain a *Mapping-AUI2CUI* model. This allow us to go through the transformation chain and, for instance, retrieve the source task from which a button has been generated.

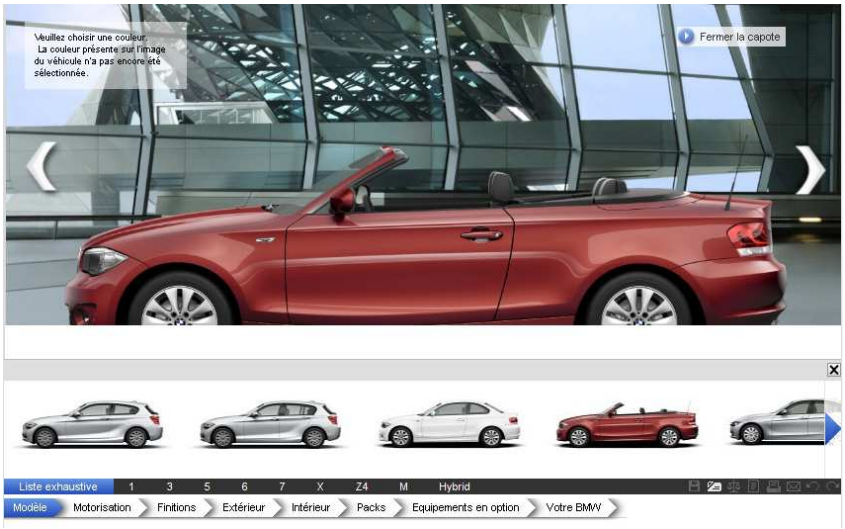

Fig. 1. Screenshot of the prototype.

Figure 1 shows an excerpt of the UI of the prototype. The UI is divided into two main parts. A big area in the middle and a thin area at the bottom. The central area of the UI has two different roles. On one hand it serves as a visual feedback for the user when

he/she selects a car model or changes the color of the vehicle (figure 1). On the other hand, it can show dialogs containing all the possible options that the user can select to configure the car with. The thin area at the bottom allows users to navigate through several categories of options for accessing different features of the car such as the electronic equipment or the exterior color of the vehicle.

The prototype was build according to the approach described in [4], so the different types of questions previously discussed could be generated and fully integrated into the UI of the prototype. The infrastructure (figure 2) consists in two model-based UIs, the self-explanatory facility for providing the help and the application. For a discussion on how to mix both set of models see [4]. 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.

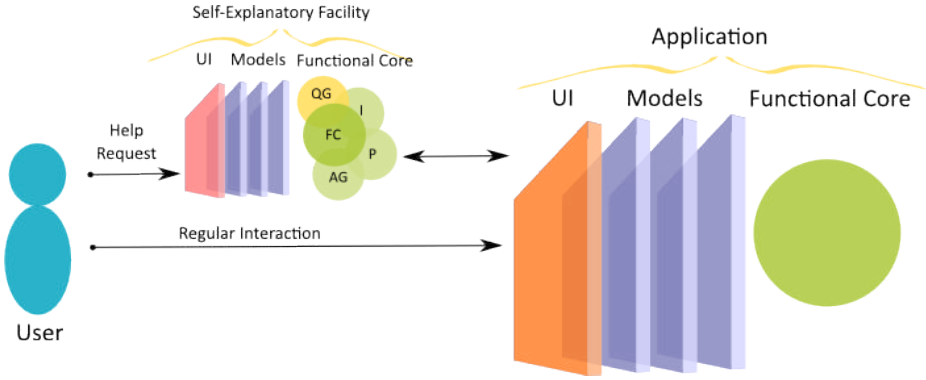

Fig. 2. Infrastructure for self-explanatory UIs. The possible questions are generated by the Question Generator (QG) from the Functional Core of the help facility (FC). For each user's request, the Interpreter (I) determines its type and parameters, used by the Processor (P) to compute the answer, which 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.

4.2 Self-Explanation Infrastructure and Questions / Answers Computation

To check the added-value of model-driven UIs in terms of explanation purposes, we supported six different types of questions built all of them upon the main models of the Cameleon Reference Framework. Figure 3 summarizes the models that have been used to generate the questions and their respective answers. For each of the types, we explain the algorithm we followed to generate the questions, the algorithm for the answers, and the involved models used in each case (summarized in figure 3).

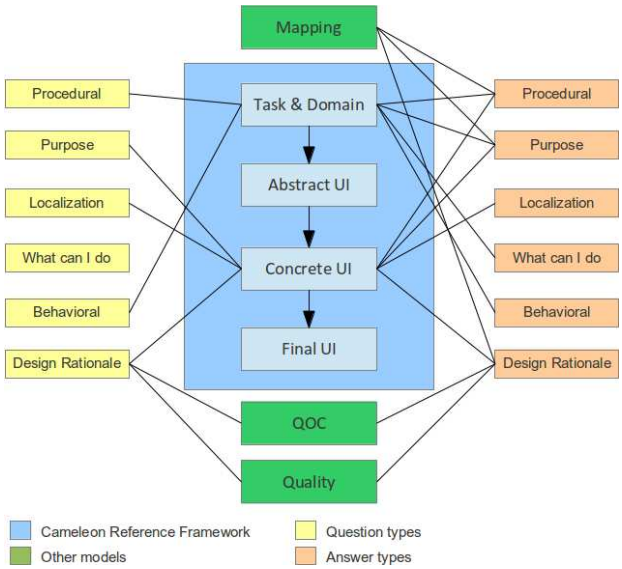

Fig. 3. Models used for generating questions (left) and answers (right).

4.3 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* + ? [1]

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

How to choose Packs?

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 ele-

[1] Original questions were asked in french. The shown grammar as well as the examples and their related answers are adaptations from the original questions presented in this research.

ments 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

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 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.

4.4 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.

4.5 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 didn't want the user to describe such widgets and thus, asking open questions that the system couldn't 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

4.6 What Can I Do Questions

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. We used the CTT notation to describe the task model so we find the available tasks as follows. 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.

4.7 Behavioral questions – Why

Behavioral questions were generated under the form:

Why I can't + *task-N.name* + ?

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

Why I can't 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]*

4.8 Design Rational Questions

We also included a QOC and a quality model to compute questions and answers about the design rationale of the UI. A detailed description of the models and the procedure

can be found in [6]. The proposed questions were directly retrieved from the QOC model. This model is a simplified version of the one in [6]. 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'.

4.9 Self-explanatory dialogue

The questions were presented in a textual form inside a dialog box (figure 4). Textual answers shown up after clicking on the desired question. In the experiment, questions were presented one by one and only at the end all the questions were shown together. We didn't filter out any question in this dialog, i.e., all the possible questions that the system was able to answer were proposed to the user. The reason for this was to show the user all the questions, so he/she can better realize if the self-explanatory system could cover his/her expectations for the given type of question. For instance, if the user realizes that his/her question is missing in the list.

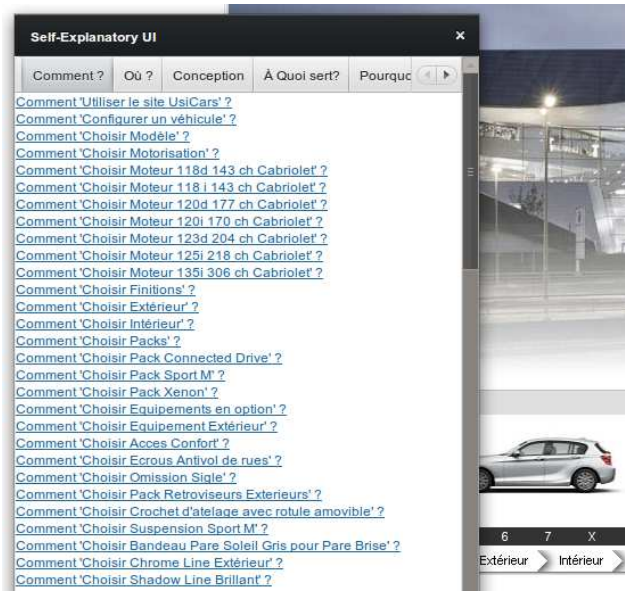

Fig. 4. Self-Explanatory Dialog box showing the full list of available questions by type.

5 User Study

We conducted an experiment to evaluate the added value of model-based self-explanations. This section starts describing the participants involved in the experiment. Then it describes each of the different phases that integrate the evaluation protocol. Last section details the prototype that we have developed to carry out the study.

5.1 Participants

We selected 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 their experience with interactive systems because the possible added value of model-based explanations can vary regarding the experience of each profile.

5.2 Method

To carry out the study, we broken-down the evaluation protocol into three different phases.

In the first phase, we asked the participants to answer a questionnaire. This questionnaire allowed us to better know the background of participants, to understand their habits regarding how they use new technologies in general, what are their common uses, the kind of applications they use with a relevant frequency, the problems they use to find with these or other applications, as well as their habits for solving these problems. The questionnaire also included questions regarding how participants used the help provided by the applications they use, and how they used to proceed in case they have a problem with the application. A software recorder was used to record all the answers of all the participants.

In the second phase of the experiment, we asked the participants to use the prototype that we had developed to this aim. We asked them to complete 10 different tasks in an established order. All the participants received the identical 10 tasks. We randomized the order of the tasks for each participant to avoid side effects such as the influence between different tasks or memory effects that can help users to accomplish the tasks better in a certain order. This part of the experiment was conducted on a laptop and the audio was recorded. We asked the participants to verbalize their thoughts, specially the questions they would like to ask to the system and the problems that they find when accomplishing the tasks.

The third part of the experiment presented the prototype including a self-explanatory dialog that contained one type of question at a time. The six questions discussed previously 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 the third phase, all the types of

questions were shown together into the same self-explanatory dialog, and we asked some more general questions that are discussed in section 6.

5.3 Tasks

The motivation for the second phase of the experiment was to confront the users with different kinds of problems that are frequently found in UIs. To this end we designed 10 different tasks. The tasks were selected according to their complexity, ranging from easy tasks to more complex ones. We didn't force any specific problem in the tasks that could be easily solved by one of the previous questions. Instead, we tried to reproduce a realistic use case with a varied set of tasks so the answers of the participants in phase 3 were not influenced by the second phase.

The 10 different tasks that we asked the participants to complete are shown in table 1. The accomplishment ratio indicates whether the participants were able to complete the tasks at all. A few users that got stuck and required hints were counted as unsuccessful. The accomplishment ratio gives an idea of how difficult each task was, regardless the expertise of the user.

Some of the tasks involved selection with searches through small lists (1, 2, 4, 8) while others involved selection through lists having multiple options and categories (5, 7, 9) in different locations. Tasks 1, 2 and 4 involved selections through images while the rest of the selection tasks were through options in textual form. Other tasks involved verification (6, 7, 10), comparison (7), or manipulate cars related terminology that was more or less easy to understand (1, 4, 6, 8).

Table 1. List of tasks and their accomplishment ratios. The tasks were randomized to avoid side effects such as the influence between tasks or memory related effects. The accomplishment ratio give an idea of the difficulty of the task.

Task description	Accomplishment ratio
1. Select a “Cabriolet” model	20/20
2. Select a diesel engine for less than 35.000 €	17/20
3. Choose a sport finishing touch	15/20
4. Change the exterior color to *Le Mans Blau*	15/20
5. Ensure that the model has a navigation system. If not, add one.	12/20
6. Ensure that the model has a *Terra* leather upholstery. If not, choose a blue leather instead.	12/20
7. Make sure that you can listen music in the car. If not, choose the best audio system available.	12/20
8. Select the Connected Drive pack	18/20
9. Select a Maintenance Contract of your choice	10/20
10. Visualize the result and check that everything is OK. If not, try to solve the problem.	12/20

We used the accomplishment ratio in the last part of the experiment, specially when we asked the participants if they believed that the model-based explanations could help them to complete one of the problematic tasks, or doing it in a more efficient way. The next section discusses the results of the qualitative analysis that we carried out with all the collected data.

6 Qualitative analysis

A large amount of qualitative data was collected from the experiment. We extracted around three hundred comments from the records made during the second phase of the experiment, the one in which participants were asked to complete the list of tasks. The selected method of analysis was the thematic type [11]. This method is focused on the answers and comments recorded during the experiment, and classified into categories later on. The aim of the thematic analysis is to group together answers or parts of answers that have the same meaning. The thematic groups were then analyzed to identify the different categories of opinion. The objective is to gather and list all the themes covered by the answers to reflect the widest possible range of opinions, distinguishing the positive ones from the negative ones.

From the extracted comments, we identified around 250 verbatims that referenced types of questions either in an explicit or implicit way. Only those verbatims that clearly related a question type where considered. For instance, verbatims like *"I don't know where the contracts are"* were classified as an implicit question of type *Where*. The table 2 shows the results of this classification, as well as some illustrative verbatims. It is significant that most of the verbatims addressed navigational problems (Where + How) mainly due to usability issues and to the nature of the tasks (table 1). The high number of 'Other types' is mainly due to questions about semantic information relating concepts specific to the domain. These questions are described in section 6.2, while next section presents the findings for both positive and negative opinions, as well as some revealed limitations of the approach.

Table 2. Relationship between question types and ocurrences extracted from the records during the second phase of the experiment. An exemple of verbatims illustrates each type.

Question type	Example Verbatims	Occurrences
How	*I don't see how to do it*	13
Why	*Why do I need to register?*	21
Where	*And where do I find the maintenance contracts*	119
What is it for	*I'm browsing the tabs to see what they do*	7
What can I do now	*I must find my way (inspecting all the UI with the mouse)*	2
Design rationale	*Why they are not ordered by type?*	1
Other types	*What does Cabriolet stand for? (Definition)*	81
	What are the differences between the packs? (Differences)	
	Is it included in the price I guess? (Confirmation)	
	What happens if I click here? (What if)	

6.1 Findings

In the first phase of the study we collected the data described in the previous Participants section, and we also found that 16/20 liked new technologies, 17/20 use new technology everyday, and 20/20 have found problems in their use. To face these problems, 11/20 inspect the UI to try to solve them by themselves, 8/20 ask other people about the problem, 15/20 search for solutions in the Internet, and 7/20 use the help provided by the system.

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 can be very useful in certain situations*" or "*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 with statements such as "*It is a gain of time*" or "*it makes me go faster without losing my time*". 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 didn't 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 didn't show a good acceptation by the participants as in the results found by [9, 10]. This was due to the fact that the questions proposed by our algorithms didn't cover all the possible range of questions that the participants asked. For instance, as our algorithms entirely rely on the task model, our system couldn't answer why questions concerning the functional core of the application such as *Why there is no diesel engines?* (for specific kinds of Cabriolet cars). To overcome this limitation in our implementation, we propose to enrich the Cameleon models with other models more suitable for this purpose, as for instance those used in [10, 20]. Another possible solution is to enrich the mapping model to include the ECA rules used in [20], so we can connect the methods of the widgets triggering the action directly to the functional core and vice versa.

Finally, the *What can I do now* and design rationale related questions were found to be useless by most of the participants (16/20), according to statements such as "*I don't see where I would like to ask this question*" (for *what can I do now?)* or "*I am not interested in this information, all I want is to buy my car*" (for design rationale questions).

At the end of the third phase, 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. Next section discusses the possible model-based implications for the types of questions that were not supported. Then, we

discuss the usability suggestions extracted from the data for our particular implementation of self-explanatory UIs.

6.2 Unsupported types of questions

We identified other types of questions not explicitly supported by our system. 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 don't know the consequences of an action but they perform such action on the UI anyway to see what happens), 2 users out of 20 didn't use options from the UI because they didn't know their possible side-effects. For instance, subject 9 didn't perform one of the tasks because *"I have fear of losing 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. These answers will probably require some improvements for side effects related to the functional core of the application (external to the UI).

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. Supporting questions about confirming and validating the user actions can help to overcome this usability issue. This may require new models for handling user actions, specially those that have effects beyond the UI.

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 between the packs?* (or eventually similarities).

6.3 Usability Suggestions and Improvements

We were also interested in usability observations. During the third 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" ([1,16]), and "Help should not display irrelevant information" ([7]).

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 don't 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. For instance, as the CUI model can store the screen coordinates of the widget, *Where* questions can be answered by highlighting the region of interest (as currently done in mac systems), and procedural questions can be explained by means of animations of the cursor over the widget coordinates.

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?

7 Conclusions and Perspectives

Model-Driven Engineering of UIs has been extensively used for the automatic generation and adaptation of UIs. This paper studies a side effect of keeping these models at runtime. They can be used for supporting the users through explanations based on such models. These explanations have interesting advantages as the dynamic generation of answers and the automatic evolution along with the models, so no manual updates of the support are required when the program specification (the models) changes. But do users think that this model-based support is relevant and useful enough? The experiment that we conducted shows that most of the users identifies model-based explanations as potentially useful.

Our future work 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 the new types of questions that we have identified, 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.

8 Acknowledgments

This work has been funded by the UsiXML. We warmly thank all the participants that collaborated in this experiment. Note that all the content used in the prototype, extracted from the original website during the reverse engineering process, belongs to this website and has been used for research purposes only.

References

1. Dix, A. Finlay, J. Abowd and G. Beale, R. Human Computer Interaction. London: Prentice Hall. (1993)
2. Calvary G., Coutaz J., Thevenin D., Limbourg Q., Bouillon L. and Vanderdonckt J. A unifying reference framework for multi-target user interfaces. Interacting With Computers Vol. 15/3. pp. 289-308. (2003)
3. Eisenstein J., Rich, C. Agents and GUIs from Task Models. Information Science. (2002)
4. 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)
5. García Frey, A., Calvary, G. and Dupuy-Chessa, S. Xplain: an editor for building self-explanatory user interfaces by model-driven engineering. In Proceedings of EICS'10, ACM, New York, NY, USA. pp. 41-46. (2010)
6. García Frey, A., Ceret, E., Dupuy-Chessa, S. and Calvary, G. Quimera: a quality metamodel to improve design rationale. In Proceedings of EICS'11, ACM, New York, NY, USA. pp. 265-270. (2011)
7. Horton, W. Designing and Writing On-line Documentation (2nd ed). New York: John Wiley & Sons. (1994)
8. Lim B. Y. and Dey A. K. Assessing demand for intelligibility in context-aware applications. In Proceedings of Ubicomp'09, pp. 195-204. ACM. (2009)
9. 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. (2009)
10. 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)
11. Paillé, P. and Mucchielli, A. L'analyse qualitative en sciences humaines et sociales. Armand Colin. Paris. (2003)
12. 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)
13. 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)
14. 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)
15. Sellen, A. and Nicol, A. Building user-centred on-line help. In R. Baecker, J. Grudin, W. Buxton, and S. Greenburg Eds. Readings in Human Computer Interaction (2nd ed) San Francisco, Morgan Kautmann. (1995)
16. Shneiderman, B. Designing the User Interface (2nd ed). Addison-Wesley. (1992)
17. Spieker P. Natürlichsprachliche Erklärungen in technischen Expertensystemen. Dissertation, University of Kaiserslautern (1991)
18. 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)
19. Vanderdonckt, J. Model-driven engineering of user interfaces: Promises, successes, failures, and challenges. In Proceedings of ROCHI'2008. Iasi, Bucarest. pp. 1-10. (2008)
20. Vermeulen, J., Vanderhulst, G., Luyten, K. and Coninx, K. PervasiveCrystal: Asking and answering why and why not questions about pervasive computing applications. In Proceedings of IE'10, IEEE Computer Society, Washington, DC, USA. pp. 271-276. (2010)