University of Castilla-La Mancha

User-Centered Reverse Engineering

Francisco Montero, Víctor López-Jaquero, Pascual González
Laboratory of User Interaction and Software Engineering
University of Castilla-La Mancha
Campus, s/n
02071 – Albacete (SPAIN)
{ fmontero | victor | pgonzalez}@dsi.uclm.es
+ 34 967 59 92 00

Albacete, 15th March 2013

User-Centered Model-Based Reverse Engineering

Francisco Montero, Víctor López-Jaquero, Pascual González

Laboratory of User Interaction and Software Engineering
Computing Systems Department
University of Castilla-La Mancha
Albacete 02071 - Spain

{victor,enavarro,fmontero,pgonzalez}@dsi.uclm.es

Abstract. User interface development introduces many challenges, arising from the different interaction styles, the diversity of both users and platforms, and the diverse contexts of use. Many of these challenges have been discussed in different papers, and they are supported by different tools aiming at supporting the so called forward engineering. The most prominent approaches proposed so far to deal with these challenges fall into two design philosophies: model-based user interface development environments (Mb-UIDE) and user-centered design (UCD). Although many efforts have been devoted to forward design under these two philosophies, less effort has been spent in the consideration of the development of user interface backwards, that is, using reverse engineering. Using a backward development path in user interface development introduces some advantages, especially when dealing with legacy systems, where the developer has to migrate pre-existing systems where the code is not currently available or where the system has to be ported to a different platform or interaction style in an agile manner. Although the widely-accepted user interface development framework CAMELEON considers both forward and reverse development, reverse one can be further refined to provide a better guidance to the developers. This paper is aimed at providing some extra guidance to those user interface developers interested in applying reverse engineering and UCD by describing the users, roles and task, identifying the specification constructs required, processes, information requirements and modeling.

Keywords: model-based user interface development, user interface development, reverse engineering, user-centered design.

1 Introduction

Technology is continuously evolving at a very quick pace. This evolution leverages the many different computing platforms which appear, and it fosters the development of novel interaction approaches. This is actually very exciting for the users of these technologies, as they are getting more and better solutions to support their activities.

Nevertheless, it has also become a real challenge for development industry. Trying to develop for the many platforms currently available, and prepare their developments for future platforms is a complex task. Also the continuous evolution of programming languages is another important issue that must be considered to provide valid solutions to this issue.

One solution to tackle this problem is using frameworks that support a wide range of platforms, such as UNITY [33]. Despite these frameworks help in addressing the issue, they are not driven or based on models, making them hard to be used in big projects. Furthermore, they do not solve the problem of legacy systems already developed under different frameworks. On the other hand, the support for future platforms depends on the developers of the framework to update it.

Current software trends move towards model-driven or model-based approaches that use models as first level entities used to generate the product, instead of just using the models to document and guide the development. This is a trend not just for general software development, but also for user interface development [20]. Most efforts in the research in model-driven development have been focused on the so called forward development, where an artifact is generated out of a more abstract one. Less effort has been devoted to the opposite path, that is, star from the artifact to extract the source models, code or documentation. Reverse engineering [6], whose aim is to analyze a system in order to identify its current components for creating a meaningful abstract model of the system, is especially useful for legacy applications, since often the original design, and sometimes the original code, might not be available, the company that made had disappeared or we might need porting the code to a new platform.

Different uses have been identified for reverse engineering [6], including restructuring, reengineering or design recovery. These terms can be adapted to user interface development. Thus, an example of restructuring in user interface development is when some widgets in the user interface are replaced with some others. An example of reengineering would be reverse engineering an application, and latter forward engineering the application to generate the application for a different target platform. Finally, an example of design recovery would be inferring the task structure of an application from the user interface.

There are several approaches aiming at reverse engineering software applications, but we can classify them into two main approaches: static and dynamic. The static one is focused on the analysis of the source code to discover the static structure of the application and extract some specific models. On the other hand, the dynamic approach examines the system at run time, simulating the actions of a user and obtaining information about its behavior. Also, as Canfora et al. [5] point out, we can find two others approaches: the hybrid and the historical ones. The first one combines the static and dynamic approaches gathering information about the static structure and the behavior. The second one includes historic information to see the evolution of the software system.

Although these proposals were previously used to gather information about the business logic aspects of the system, there are some others that try to apply the same strategies to extract information from the user interface. Thus, plenty of studies can be found on static reverse engineering by using different strategies [29,31,32]. Moreover, some of them use a model-based approach based on the CAMELEON framework [1,2,3,34], the same one that we use in our work. After analyzing all these proposals based on the static strategy, we found that all of them are tied to a specific programming language, like C++, Java, HTML, Oracle Forms and so on, being this fact an important limitation that constraints the use of those solutions. Although other authors, like Silva et al. [29], try to overcome this shortcoming by including the generation of an Abstract Syntax Tree (AST) that can be analyzed independently of the language of the source code. Nevertheless, the initial phase that creates the AST model is language-dependent.

On the other hand, there are also some approaches based on dynamic reverse engineering of user interfaces [7,12,14]. The vast majority of such proposals are aimed at testing purposes and often they include some type of static analysis.

Thus, as we have shown, in recent years user interface reverse engineering has attracted a growing interest [2], but all the proposals so far include the analysis of the source code, so their applicability is constrained to the specific languages supported by each approach. However, we can find some systems where the source code is unstructured or some solutions like SOA where the source code could not be available. Also the knowledge of the real functionality of the systems is very difficult to gather only by using the source code, so other approaches should be used.

In this paper we present and approach aimed at extracting the functionality and the structure of the system by analyzing the behavior of the system in real time, but also by applying user centered design techniques as a complementary source of information. The final goal is not to obtain just the requirements of the system, but also to gather design knowledge at different abstraction levels that can be used in latter developments or updates of the application being reverse engineered. Therefore, in this work we are aimed at contributing to improve the reverse engineering of user interfaces by providing a process to guide it.

This paper is structured as follows. First, the approaches to user interface design used as the foundations of this work are presented. Next, our method for user-centered reverse engineering is introduced. Lastly, some conclusions drawn and future work are included.

2 Approaches to user interface development

There are many different user interface development methods. Nevertheless, most of them take inspiration from two trends or philosophies. On the one hand, there are some methods inspired on the model-based approach (Mb-UIDE). The foundations, metamodels, evolution and challenges for Mb-UIDE are discussed in [20,25,27,28]. In more recent times, these model-based approaches are evolving towards model-driven user interface development (MDUI) [16,35], as a natural evolution from model-based to model-driven. These approaches are aimed at improving the industrialization in the development of user interfaces. That is, they pursue a more systematic user interface development with automatic or semi-automatic generation of the user interface. A user interface description language (UIDL) [13,30] is used as the underlying foundation to represent the models. These models are transformed to generate the user interface.

On the other hand, user interface development has been leveraged by approaches where the user in considered. Unfortunately, they have not been always used jointly with the model-based or model-driven approaches. These approaches are grouped under the Usability Engineering umbrella [21] and in particular User-Centered Design [22].

While the first group of approaches pursues industrialization in the development, this second group aims at improving the usability, quality in use and user experience of the user interfaces created.

Both groups of approaches have been focused mostly on the forward path of user interface development. That is, they start from requirements until they are accomplished in their realization.

In the next section we have included a short review of these two design strategies in which our proposal is based on.

2.1 User-Centered Design

To design user interfaces which are usable in a specific use situation, active involvement of representatives of the user population is essential. Therefore User-Centered Design (UCD) [22] is a must for designing usable systems. User-centered design is an approach for improving usability, quality in use and user experience. UCD is a structured development methodology that involves users throughout all stages of software development in order to create a software product that meets user' needs. This approach considers an organization's business objectives and the user's needs, limitations, and preferences and not especially other criteria like productivity or reutilization

Nowadays, the view on user-centered design of usable software is based on ISO international standard on human-centered design process for interactive systems [17]. But previous works, such as [11] established the principles for designing for usability, and these principles were: (1) early focus on users and tasks, (2) empirical measurement, and (3) iterative design.

There are many user interface development proposals following user-centered design principles, for instance, [10] [15] [19] [21], but all those proposals are not model-based ones. However, in [8][9] there is a clear consideration of the concept of model and user interface development process.

2.2 CAMELEON Framework

The CAMELEON Reference Framework [4] is probably the most widely-accepted user interface development framework. Actually, it is currently being used as the reference framework for Model-Based standardization in W3C [36].

The CAMELEON Reference Framework promotes a four-step forward engineering development path starting with domain concepts and task modeling. Although research in Human-Computer Interaction (HCI) has promoted the importance of task modeling, practitioners often skip this stage, and directly produce concrete user interfaces (CUIs) using prototyping tools such as *Flash* because of the lack of tools allowing rapid prototyping from task models. This practice corresponds to the last two steps of the reification process recommended in the reference framework. Nonetheless, the framework can be instantiated with the number of reification steps that fits designer's culture. In other words, designers can choose the entry point in the reification process that best fits their practice. If necessary, the higher missing abstractions in the reification process can be retrieved through reverse engineering.

In this sense, reverse engineering is a composition of abstractions and code reverse engineering enabling a transformation of a low-level viewpoint into a higher level viewpoint. Following the CAMELEON's levels (**Fig. 1**), several transformations can be identified (T1, T2 and T3 in **Fig. 1**).

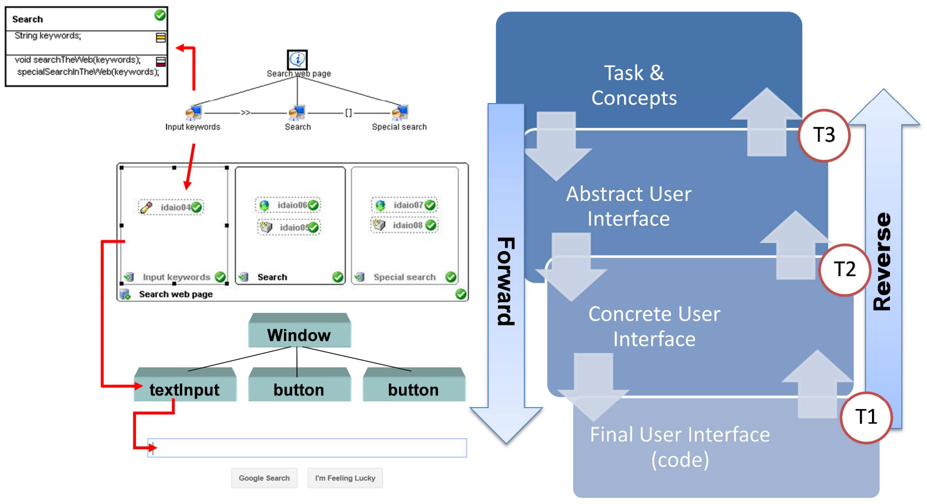


Fig. 1. CAMELEON abstraction levels.

Under reverse engineering, abstraction is an operation intended to map a UI representation from one non-initial level of abstraction to a higher level of abstraction. i.e., deriving an abstract user interface from a concrete user interface.

In the context of legacy systems, the most ambitious development path is retargeting. Retargeting is useful in processes where an existing system should be retargeted, that is, migrated from one source computing platform to another one that poses different constraints. Retargeting is a composition of reverse engineering, context adaptation or redesign and forward engineering. The source Final UI code of the legacy system is abstracted into a CUI (or an AUI). This new CUI and/or AUI are redesigned according to specific adaptation or design heuristics for the target platform. From this redesigned CUI and/or AUI specification a new interface code is created by using a forward engineering process (the so called reengineering).

Probably the instantiations of CAMELEON framework most salient are UsiXML [18] and MARIA [23], where both forward and reverse development paths are considered together with some related tools.

UsiXML (which stands for USer Interface eXtensible Markup Language) is a XML-compliant markup language that describes the UI for multiple contexts of use and modalities, including character user interfaces, graphical user interfaces, auditory user interfaces, and multimodal user interfaces. In other words, interactive applications with different types of interaction techniques, modalities of use, and computing platforms can be described in a way that preserves the design independently from particular characteristics of the context of use. UsiXML is supported by several tools, being most of them forward-oriented, although, as we have mentioned before, there are some proposal [3,34] that try to offer a solution of reverse engineering for the web applications domain.

MARIA is both a framework and a tool; it supports the description of user interfaces at abstract and concrete levels. The abstract language is independent of the interaction platform. A number of concrete languages are part of MARIA and provide a refinement of the abstract description for several target platforms (graphical desktop, graphical touch-based smartphone, graphical mobile, vocal and multimodal (combination of graphical and vocal). Again, MARIA is designed for forward development, but as in the case of UsiXML, in this environment there are some proposals [1, 2] that include reverse engineering for the web applications domain.

3 SECREM: User-Centered reverse engineering in model-based user interface design

The method SECREM (uSEr Centered Reverse Engineering Method) supports reverse engineering activities of systems, even no source code is available. It features two main contributions with regard to the other reverse engineering proposals previously discussed. First, SECREM makes extensive use of the different models used in the model-based or model-driven user interface approaches, and it does not limit to some of them. For instance, in other tools such as reversiXML[3] or ReverseAll[1] they start from HTML code, and they just consider some models at the concrete user interface level. However, SECREM considers specific users performing specific tasks by using some screenshots from the running system, and by using prototyping techniques applied to the snapshots and tagging the elements identified in the prototypes. Second, SECREM involves the user of the application being reverse engineered. To do so, the following user-centred techniques have been integrated into SECREM:

- Field or ethnographic studies can help in the identification of complex or critical design challenges. An expert is required during the observation of the target audience in a real-life environment. In user-centred design ethnography contributes to understand better the design problem. For instance, if an e-commerce system is being created for home appliances, an ethnographic study could be conducted in a physical home appliances shop to gather information about, for instance, how the customers choose between appliances.

- *Personas* method [10] aims to identify and communicate user needs efficiently and effectively. In *Personas* archetypal users are created out of the user needs gathered from real users. Thus, designers focus in designing the applications for these *personas*, rather than for single specific users.
- Prototyping is a method used by designers to acquire feedback from users about future designs. Prototypes are similar to mock-ups, but they are usually not as low-fidelity as mock-ups and appear slightly later in the design process.

In **Fig. 2** an overview of SECREM framework is depicted.

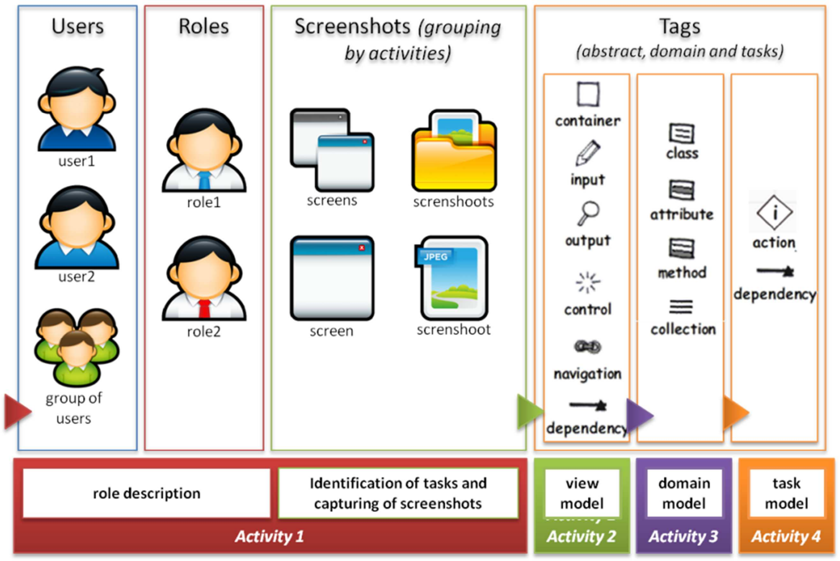


Fig. 2. SECREM framework for user-centred reverse engineering.

This figure illustrates the roles, screenshots and tags used to identify the different elements and their relationships in the prototypes elaborated starting from the screenshots. In the bottom of the figure the different models derived during prototype tagging activities in SECREM are gathered. SECREM supports the generation of conceptual models associated to the view, the business logic and the domain model. Next, how these models are automatically generated by using our tool will be explained in depth.

3.1 SECREM: the method

The input of SECREM is the system that the designer wants to apply reverse engineering. The motivation to apply reverse engineering to a system can result from many different reasons. For instance, the system might require some maintenance or updating and it is not directly possible because of the lack of documentation or source code of the system. Sometimes the system does not exist yet, but the target users of the system are available to the designers. Thus, SECREM could be used to gather information about the system-to-be by using prototyping.

As a result of applying SECREM some conceptual models will be produced automatically from the information provided during the prototyping and tagging activities carried out by the designer. These models correspond to different views of a software system. Next, these models are described:

- The *view model* represents all those entities and relationships among them of the interactors the users use to interact with the application. The view model is a conceptual model of the usual presentation models (*concrete and abstract user interface* in CAMELEON framework) used in the model-based user interface development approach.
- The *domain model* is related to the real world entities and the relationships between them. This domain model is manipulated by the user through the user interface provided by the system. The domain model is closely related to the domain model (*Concepts* in CAMELEON framework) found in the model-based user interface development approach.
- The information usually stored in the *task model* in the model-based user interface development approach is scattered in different places in our approach. The order that the user can follow to use the interactors can be specified in the prototype. On the other hand, the navigation map between screenshots can be also specified. This navigation can be between screenshots for the same activity (intra-activity) or not (inter-activity).

Moreover, SECREM includes some user-centred techniques, as aforementioned. These techniques are used in the first stages of the approach to know the users of the system SECREM is being applied to. Both the users of the system and the roles they play will be gathered by using both ethnographic studies and *Personas* method [10].

The second part when putting into practice SECREM consists in working with screenshots. Prototyping and tagging activities are applied to these screenshots to generate the models previously discussed. Exactly, the following activities are applied to the screenshots:

- ***Activity 1*** (***A1 - Working with users***). At first, designers and users should work together. Designer talk to representative users to gather information regarding both the users and their tasks. They, both designers and users, talk about the tasks that users perform through the system and how they do them. The designers will capture and organize some screenshots for each task. Each set of screenshots is linked to a specific task the user can carry out in the system, and meaningful for him. Designer can gather information about users; such as role, preferences and needs. The results of this activity are related to the information about the users and their activities.
- ***Activity 2*** (***A2 – Abstracting user interface***). After those information and screenshots are organized, they are loaded into a tool supporting SECREM. An abstract interaction object is associated to every interactor shown in the screenshot. This

specification is based on prototyping techniques. Dependencies between these interactors are also specified. These dependencies are useful to specify the behaviours within the user interface. The results of this activity are related with the view model.

- ***Activity 3*** (***A3 - Abstracting domain model***). Latter, for each organized set of screenshots the domain entities they manipulate are identified, together with the attributes of those entities referred from the different screenshots. A domain model is the main outcome of this activity.
- ***Activity 4*** (***A4 - Abstracting task model***). At the same time, it is also possible to specify the intra-container order, that is, the order the user is supposed to interact with the abstract interactors in each screenshot. Optional abstract interactors and dependencies between abstract interactors and containers are also specified at this stage. A task model is the more relevant outcome of this activity.

In SECREM method the conceptual models aforementioned (view, domain and tasks) will be generated while the previous activities are carried out. Next, these activities will be explained in detail to show the benefits of using SECREM approach.

A1. Working with users

The first activity considered in SECREM method is the identification, selection and knowledge of the users of the application currently available, and that is going to undergo the reverse engineering process. The knowledge about those users will help in finding out what their impression of the application is and observe how they use the application. Ethnographic studies contribute to answer the following questions:

- Who are the users of the application?
- Which roles do they play in the application?
- What are the main activities the users perform in the application?
- How do they carry out their activities in the application?
- What are the common features of the users of the application?
- What do they think about the way they currently carry out their activities?
- Do the users exhibit some special requirements that should be considered?
- What are the goals of the users or what do they use the application for?
- What activities are more important for the users?

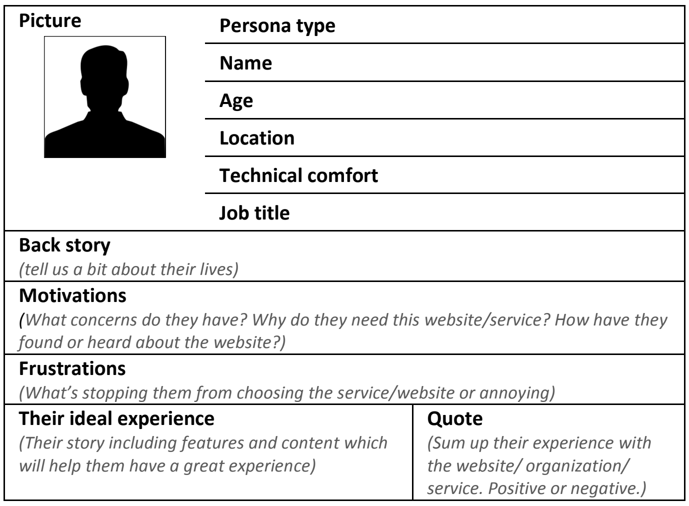
The information gathered in this stage is textual, by using templates to identify the users and their activities.

For the designers is especially useful the information gathered during this activity to know particular or extra features of the application and its usage. Moreover, this activity will enable setting a priority in the specification of the activities and to detect different user views of how an activity of the application is carried out.

To gather the aforementioned information, besides knowing the users, the designer should document the roles that those user play in the application. To document those roles the method *Personas* [10,26].

The template used to document the users and roles is shown in **Fig. 3**. It takes inspiration from the one proposed by OrangeBus (http://www.orangebus.co.uk/).

As a result of this activity the designer should be ready to capture snapshots for each activity the user carries out. These snapshots, organized according to the activity they are used in, are the input for the second activity in SECREM method. Next, this second activity will be described.



Picture	**Persona type**
	Name
	Age
	Location
	Technical comfort
	Job title
Back story *(tell us a bit about their lives)*	
Motivations (*What concerns do they have? Why do they need this website/service? How have they found or heard about the website?)*	
Frustrations *(What's stopping them from choosing the service/website or annoying)*	
Their ideal experience *(Their story including features and content which will help them have a great experience)*	**Quote** *(Sum up their experience with the website/ organization/ service. Positive or negative.)*

Fig. 3. Identifying users and roles.

A2. Abstracting user interface

In this activity each activity starts with a set of screenshots from the application being reverse engineered. These screenshots are analyzed and annotated by using a set of tags related to abstract user interface modeling, conceptual domain model and task model.

Table 1. The tags used to annotate the abstract user interface screenshots in SECREM.

Icon	Name	Description
☐ container	Container	Composition of one or several abstract elements of interaction (input, output, control and navigation).
✎ input	Input	Representation of an abstract element enabling the user to enter data through the interface.
🔍 output	Output	Representation of an abstract element enabling the system to output data through the interface.
✳ control	Control	Representation of an abstract element enabling the user to trigger some behaviour in the interface.
∞ navigation	Navigation	Representation of an abstract element enabling the user to navigate through the interface.
⟶	Dependency	Dependency is a relationship that means that a single or a set of

		abstract elements notifies to and listens to other abstract element for their specification or implementation.
≡	Collection/ repetition	It represents a collection of abstract elements.

The tags available in SECREM for the designer are shown in the left part of **Fig. 2**. These tags are further described in **Table 1**.

Another set of tags is used in SECREM to identify and specify in the screenshots the entities, attributes and methods manipulated by the user interface. These annotations are used to derive the domain model of the application. Obviously, the entities manipulated through the user interface cannot be derived from a single screenshot or set of screenshots, since it would offer a partial view. Therefore, SECREM tool supports the designer in the specification of additional attributes or even new entities. Furthermore, since the same entity could appear as a result of several screenshots an integration process to merge different entities derived from the screenshots is also supported. The tags used in SECREM to annotate the screenshots to derive the domain model are shown in **Table 2**.

Table 2. Tags and icons for the specification of domain model in SECREM.

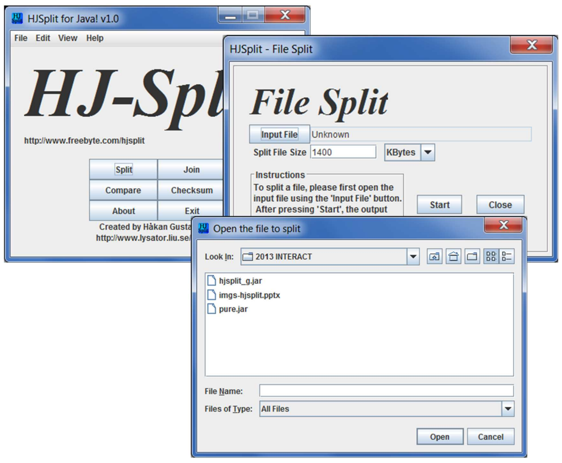
Icon	Name	Description
▤	Entity	Entity representing a piece of data retrieved from the Domain Model. An entity is a description of the classes of objects manipulated by a user while interacting with a system.
▬	Attribute	An attribute is a logical data value of an entity.
▬	Method	A method is some behaviour of an entity.

In **Table 3** illustrates the tag used to specified the order that helps in the abstracting the task model in SECREM. This tag enables the specification of the order among the set of abstract user interface elements (see **Table 1**) included in a container. The task model is also abstracted from the navigation between containers derived from their interdependencies.

Table 3. Entities and icons for the specification of the task model in SECREM.

Icon	Name	Description
◇	Order	The order of the abstract elements on a container determines the sequence in which the focus will change. Usually the order is from left to right within each row of an abstract element and from top to bottom.
→	Dependency	Dependency is a relationship used to denote a relationship between abstract interactors or between abstract interactors and other screenshots.
optional	Optional	Represents when some actions in the screenshot are not mandatory or required to carry out the task.

The previous activities are applied to the screenshots gathered in the first activity. In the following sections an example will be used to describe the activities carried out by the designer, which are more closely related to the identification and specification of the tags introduced in this section.



Input File: Unknown
Split File Size: 1400 KBytes

Instructions
To split a file, please first open the input file using the 'Input File' button. After pressing 'Start', the output

Look In: 2013 INTERACT
hjsplit_g.jar
imgs-hjsplit.pptx
pure.jar
File Name:
Files of Type: All Files

Fig. 4. Final user interface for splitting a file in HJ-Split tool.

Fig. 5. Abstract prototype for splitting a file in HJ-Split tool.

Forward model-based user interface development produces some models out of some others in an automatic or semi-automatic manner. In this sense, the abstract user interface model in CAMELEON framework is generated from the domain and the task model. In SECREM this abstract model is obtained by using prototyping based on the screenshots grouped for each activity, that is, from what could be considered the final and concrete user interface level in CAMELEON. In SECREM the abstract model is not automatically generated, but the platform independent models are generated during the prototyping process. Let's see an example of the backward process proposed in SECREM.

To illustrate our method for user-centred reverse engineering an activity to split a file into several smaller parts by using HJ-Split tool (http://www.hjsplit.org/) has been chosen. This activity is carried out by interacting with the three screenshots shown in **Fig. 4**.

SECREM prescribes in this step using prototyping techniques to tag the snapshots to identify the abstract interactors. To do so, the areas in the screenshot for each abstract interactor and the tags that denote its purpose are applied to the screenshot. The output of this activity for the screenshot in **Fig. 4** in shown in **Fig. 5**.

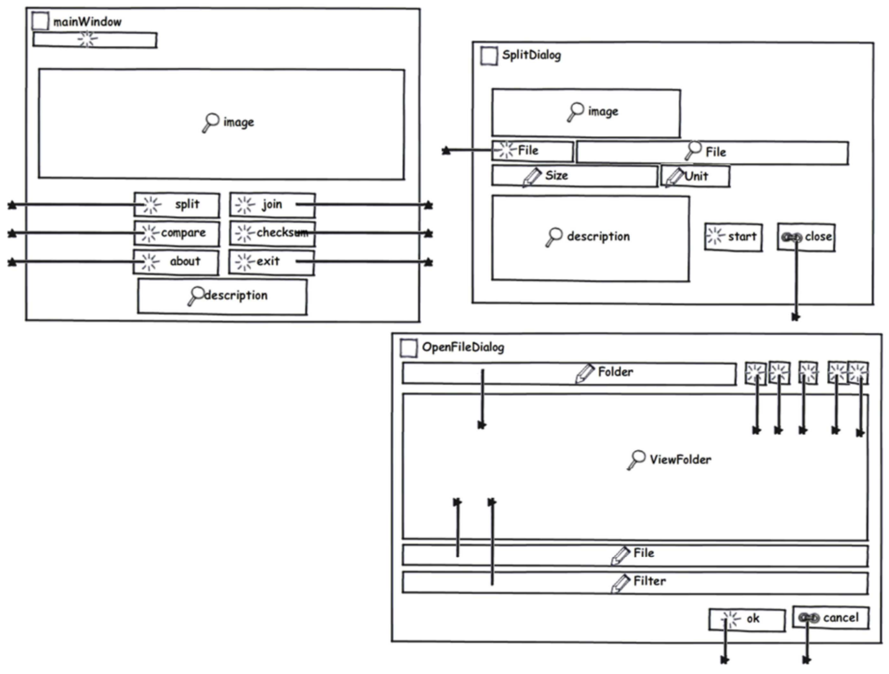


Fig. 6. Dependencies specification for splitting a file in HJ-Split tool.

The prototype in **Fig. 5** shows different abstract interactors, together with its purpose. There are elements to input information to the system, trigger some behaviour or show information to the user.

There are two types of dependencies depicted in **Fig. 6**. On the one hand there are dependencies between abstract interactors. For instance, there are dependencies between input abstract interactors and output abstract interactor. These dependencies represent that the contents of the output abstract interactor depend on the interaction of the user with the input element. A similar dependency appears between the abstract control interactors in the upper right corner of the lower prototype of **Fig. 6**. These dependencies denote that they modify what is shown in the output abstract interactor.

On the other hand, there is another type of dependency related to navigation. In the bottom right part of the lower prototype of **Fig. 6** both control abstract interactors have a dependency of this type. This type of dependency will be further explained in the section devoted to abstracting task activity.

The abstract user interface generated consists of the abstract interactors and the behaviour of the user interface. The behaviour derived automatically by generating a class diagram of the view according to the prototyping and tagging activities.

At this step, the designer and users can specify also the existing dependencies between the abstract interactors identified. As a result of this step some elements will become *notifiers* and some others will become *listeners* of those *notifiers*. An element can be at the same time a *listener* and a *notifier*. **Fig. 6** shows the dependencies specified for our example.

The conceptual model of the abstracted user interface generated is called the view model. It consists of the abstract interactors and the behaviour of the user interface. This view model and its behaviour are derived automatically by generating a class diagram of the view according to the prototyping, the tagging activities and a set of heuristics.

A3. Abstracting domain model

In this activity, designers and users can review the available view model in order to identify persistent data and the data manipulated in this view model. This information is also modelled in SECREM. The model for the data manipulated is called domain model. This activity is carried out by tagging the containers with entities, attributes and methods related with the information manipulated by users though the user interface.

The domain model consists of a set of entities and its relationships. These entities are mainly manipulated by users using the user interface of a software product. In our example, the domain model consists of two entities: *file* and *folder*. These entities are associated to the input and output files.

A4. Abstracting task model

Lastly, the information usually stored in the task model will be abstracted. The task model is, together with the previous model, one of the platform independent models essential for the model-based development of user interfaces. It represents the task the user will carry out in the system, including the organization of those tasks and the spatial-temporal relationships among the tasks. There are different notations to represent the task model, being the most widely-accepted ConcurTaskTrees (CTT) [24]. In SECREM the task model is generated from four subactivities applied to the prototyped screenshots:

- The first subactivity is the specification of the order the user is supposed to use the different interactors in the prototype for a given user interface. This is carried out by tagging the interactors with a number to represent the order (see **Fig. 7**).

- The second subactivity is the specification of the navigation relationships between the different screenshots the user visits to carry out an activity in the system (intra-activity navigation). This kind of navigation includes, but is not limited to, error messages or confirmations, that is, they are navigations within the same activity. In **Fig. 7** two examples of this kind of relationship are shown in the interactors tagged with the order numbers 3 and 4 in the lower prototype image. When the designer is creating this kind of relationship, he is specified sequential spatial-temporal relationships, as the ones used in CTT.
- The third subactivity is related to inter-activity navigation specification. That is, it is related to the specification of the navigation between screenshots from different activities.
- Lastly, the fourth subactivity is aimed at the specification of what tasks in each screenshot are optional.

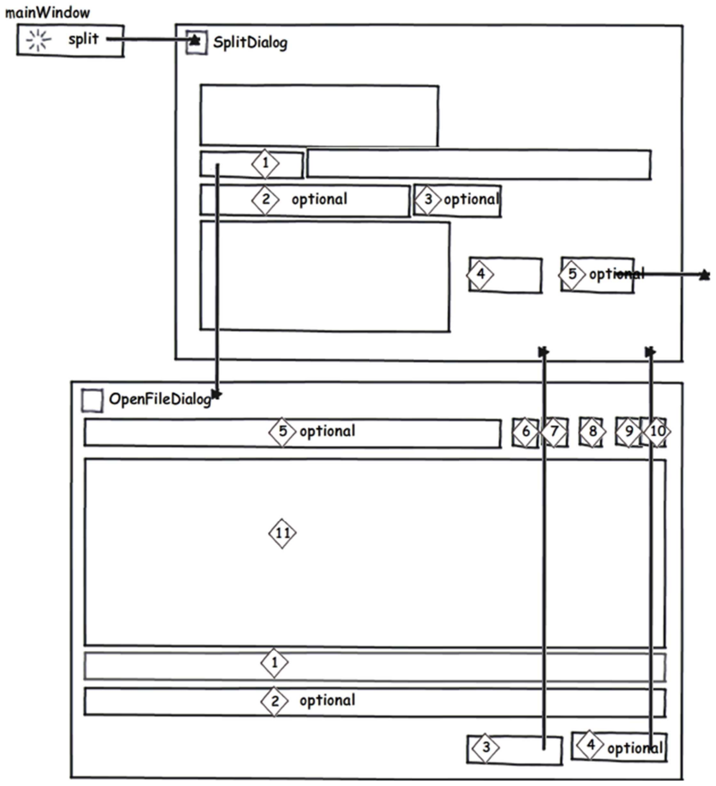


Fig. 7. Sequential order specification of interactors and navigation dependencies.

4 Conclusions and future work

In this paper a reverse engineering method, namely SECREM is described. This method puts together User-Centred Design and Model-Based User Interface Development approaches to improve current reverse engineering techniques. By including UCD in the reverse engineering process we aim at providing a dynamic model-based approach to reverse engineering, where the users and the users interacting with the running system are involved in the process to abstract information regarding the design of the user interface and the behaviour of the system. To integrate UCD in the reverse engineering life cycle some UCD techniques are used. Ethnographic studies and *Personas* method are used to achieve one of the most important tasks in UCD: knowing the users and their tasks. Prototyping is also used to guide the generation of the models abstracted from the screenshots of the running system. Using prototypes introduces very interesting advantages. Because they are simple, a user with no software design expertise can use and understand the prototypes created.

The goal of the approach is to provide a solution to reverse engineering when the source code is not available to apply other reverse engineering approaches. On the contrary to most reverse engineering approaches, another result from this method is that it is language independent. We don't rely on the source code as an input. Therefore, it can be applied regardless of the language the application was developed.

Now we are working in the development of a software tool that will offer support to all the activities described for this process. This tool will support decomposing the old system by creating three views to describe in different detail and in distinct abstraction levels the information abstracted during the reverse engineering process. The models generated from these three views can be used afterwards to, for instance, reengineer the original system or for documenting purposes.

Acknowledgements. This work has been partially supported by the grants insPIre (TIN2012-34003) and AVANZA TSI-020400-2011-20 from the Spanish Government.

References

1. Bandelloni, R., Paternò, F., and Santoro, C. Reverse Engineering Cross-Modal User Interfaces for Ubiquitous Environments. In J. Gulliksen, M.B. Harning, P. Palanque, G.C. Veer and J. Wesson, eds., *Engineering Interactive Systems.* Springer Berlin Heidelberg, Berlin, Heidelberg, 2008, 285 – 302.
2. Bellucci, F., Ghiani, G., Paternò, F., and Porta, C. Automatic reverse engineering of interactive dynamic web applications to support adaptation across platforms. *Proceedings of the 2012 ACM international conference on Intelligent User Interfaces - IUI '12,* ACM Press (2012), 217.
3. Bouillon, L., Limbourg, Q., Vanderdonckt, J. Michotte, B. Reverse Engineering of Web Pages Based on Derivations and Transformations. *Third Latin American Web Congress (LA-WEB'2005)*, IEEE, 3–13.
4. 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 15*, 3 (2003), 289–308.
5. Canfora, G., Penta, M. Di, and Cerulo, L. Achievements and challenges in software reverse engineering. *Communications of the ACM 54*, 4 (2011), 142.

6. Chikofsky, E.J. and Cross, J.H. Reverse engineering and design recovery: a taxonomy. *IEEE Software 7*, 1 (1990), 13–17.
7. Coimbra Morgado, I., Paiva, A., Pascoal Faria, J. Dynamic Reverse Engineering of Graphical User Interfaces. *International Journal On Advances in Software, 5*, 3-4 (2012), 224–236.
8. Constantine, L. and Lockwood, L.A.D. *Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design*. Addison-Wesley Professional, 1999.
9. Constantine, L.L. and Lockwood, L.A.D. Usage-centered engineering for Web applications. *IEEE Software 19*, 2 (2002), 42–50.
10. Cooper, A., Reimann, R., and Cronin, D. *About Face 3: The Essentials of Interaction Design*. Wiley, 2007.
11. Gould, J.D. and Lewis, C. Designing for usability: key principles and what designers think. *Communications of the ACM 28*, 3 (1985), 300–311.
12. Grilo, A., Paiva, A., and Faria, J. Reverse engineering of GUI models for testing. *Information Systems and Technologies (CISTI),* IEEE Computer Society (2010), 1–6.
13. Guerrero, J., González-Calleros, J.M., Vanderdonckt, J., and Muñoz Arteaga, J. A Theoretical Survey of User Interface Description Languages: Preliminary Results. *Proc. of LA-WEB/CLIHC 2009*, (2009), 36–43.
14. Hackner, D.R. and Memon, A.M. Test case generator for GUITAR. *Companion of the 13th international conference on Software engineering - ICSE Companion '08*, ACM Press (2008), 959.
15. Holzinger, A. Usability engineering methods for software developers. *Communications of the ACM 48*, 1 (2005), 71–74.
16. Hussmann, H., Meixner, G., and Zuehlke, D., eds. *Model-Driven Development of Advanced User Interfaces*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.
17. ISO. *ISO 9241-210:2010, Ergonomics of human-system interaction -- Part 210: Human-centred design for interactive systems*. 2010.
18. Limbourg, Q., Vanderdonckt, J., Michotte, B., Bouillon, L., and López-Jaquero, V. Usixml: A language supporting multi-path development of user interfaces. *Engineering Human Computer Interaction and Interactive Systems*, (2005), 200–220.
19. Mayhew, D.J. *The Usability Engineering Lifecycle: A Practitioner's Handbook for User Interface Design*. Morgan Kaufmann Publishers, 1999.
20. Meixner, G., Paternò, F., and Vanderdonckt, J. Past, Present, and Future of Model-Based User Interface Development. *i-com 10*, 3 (2011), 2–11.
21. Nielsen, J. *Usability Engineering*. Morgan Kaufmann Publishers Inc, San Francisco, USA, 1993.
22. Norman, D.A. and Draper, S.W. *User-Centered System Design: New Perspectives on Human-Computer Interaction*. Lawrence Earlbaum Associates, Hillsdale, NJ, 1986.
23. Paterno', F., Santoro, C., and Spano, L.D. MARIA. *ACM Transactions on Computer-Human Interaction 16*, 4 (2009), 1–30.
24. Paternò, F. *Model-Based Design and Evaluation of Interactive Applications*. Springer-Verlag, 1999.
25. Pinheiro, P. User Interface Declarative Models and Development Environments : A Survey. *Proceedings of the 7th international conference on Design, specification, and verification of interactive systems (DSV-IS'00)*, Springer-Verlag (2001), 207–226.
26. Pruitt, J. and Adlin, T. *The Persona Lifecycle: Keeping People in Mind Throughout Product Design*. Morgan Kaufmann, 2006.
27. Puerta, A.R. A model-based interface development environment. *IEEE Software 14*, 4 (1997), 40–47.

28. Schlungbaum, E. *Model-based User Interface Software Tools Current state of declarative models*. 1996.
29. Silva, J.C., Silva, C.C., Gonçalo, R.D., Saraiva, J., and Campos, J.C. The GUISurfer tool. *Proceedings of the 2nd ACM SIGCHI symposium on Engineering interactive computing systems - EICS '10*, ACM Press (2010), 181.
30. Souchon, N. and Vanderdonckt, J. A review of XML-compliant user interface description languages. *Proceedings of the 10th International Workshop on Interactive Systems. Design, Specification, and Verification: DSV-IS 2003*, Springer (2003), 377–391.
31. Staiger, S. Reverse Engineering of Graphical User Interfaces Using Static Analyses. *14th Working Conference on Reverse Engineering (WCRE 2007)*, (2007), 189–198.
32. Sánchez Ramón, Ó., Sánchez Cuadrado, J., and García Molina, J. Model-driven reverse engineering of legacy graphical user interfaces. *Proceedings of the IEEE/ACM international conference on Automated software engineering - ASE '10*, (2010), 147.
33. Unity Technologies. Unity3D Game Engine. 2013. http://unity3d.com/.
34. Vanderdonckt, J., Bouillon, L., Souchon, N., Louvain, U. De, and Doyens, P. Flexible Reverse Engineering of Web Pages with V AQUISTA. *WCRE "01 Proceedings of the Eighth Working Conference on Reverse Engineering (WCRE"01)*, IEEE Computer Society Press, Los Alamitos (2001), 241–248.
35. Vanderdonckt, J. A MDA-Compliant Environment for Developing User Interfaces of Information Systems. *Proc. of 17 th Conf. on Advanced Information Systems Engineering CAiSE'05*, Springer-Verlag (2005), 13–17.
36. W3C. Model-Based UI XG Final Report. 2010. http://www.w3.org/2005/Incubator/model-based-ui/XGR-mbui-20100504/.