Using Storyboards to Integrate Models and Informal Design Knowledge

Peer-reviewed author version

HAESEN, Mieke; VAN DEN BERGH, Jan; MESKENS, Jan; LUYTEN, Kris; Degrandsart, Sylvain; Demeyer, Serge & CONINX, Karin (2011) Using Storyboards to Integrate Models and Informal Design Knowledge. In: Hussmann, Heinrich & Meixner, Gerrit & Zuehlke, Detlef (Ed.) Model-Driven Development of Advanced User Interfaces, p. 87-106..

DOI: 10.1007/978-3-642-14562-9_5
Handle: http://hdl.handle.net/1942/11802

Using Storyboards to Integrate Models and Informal Design Knowledge

Mieke Haesen, Jan Van den Bergh, Jan Meskens, Kris Luyten, Sylvain Degrandsart, Serge Demeyer, and Karin Coninx

Abstract Model-driven development of user interfaces has become increasingly powerful in recent years. Unfortunately, model-driven approaches have the inherent limitation that they cannot handle the informal nature of some of the artifacts used in truly multidisciplinary user interface development such as storyboards, sketches, scenarios and personas. In this chapter, we present an approach and tool support for multidisciplinary user interface development bridging informal and formal artifacts in the design and development process. Key features of the approach are the usage of annotated storyboards, which can be connected to other models through an underlying meta-model, and cross-toolkit design support based on an abstract user interface model.

1 Introduction

[1] The last few years, model-driven design of User Interfaces (UIs) is receiving an increasing amount of attention in the computer science and software engineering community. Model driven UI design can offer benefits in terms of quality, traceability, efficiency and consistency. However, this discipline has not been widely adopted

Mieke Haesen, Jan Van den Bergh, Jan Meskens, Kris Luyten and Karin Coninx
Hasselt University - tUL - IBBT, Expertise Centre for Digital Media, Wetenschapspark 2, 3590 Diepenbeek, Belgium, e-mail: {mieke.haesen, jan.vandenbergh, jan.meskens, kris.luyten, karin.coninx}@uhasselt.be

Sylvain Degrandsart · Serge Demeyer
Dept. Mathematics and Computer Science, Universiteit Antwerpen, Middelheimlaan 1, 2020 Antwerpen, Belgium, e-mail: {Sylvain.Degrandsart, Serge.Demeyer}@ua.ac.be

Sylvain Degrandsart,
University of Mons - UMONS, Software Engineering Lab, 20 Place du Parc, 7000 Mons, Belgium

[1] Published by Springer, accessible on http://www.springer.com/computer/swe/book/978-3-642-14561-2

in the field of Human-Computer Interaction (HCI). A major reason for this is that existing model-driven UI design approaches are not focused on UI design in multidisciplinary teams. Since this is the most common way UIs are created, model-driven approaches have to fit into an overall multidisciplinary design approach to become usable for HCI practitioners.

Fitting within an overall multidisciplinary design approach means that artifacts from other communities have to be incorporated into the model-driven design approach. Some challenges have already been tackled. User interface sketches can already be integrated into a model-driven approach, based on sketch recognition to recognize widgets within a sketch [1]. Furthermore, tools like Microsoft Expression Blend support the transition from early prototypes (SketchFlow) to final designs and integration with software development (through Visual Studio). But designers and other people involved in a user interface development process also use other, often informal, artifacts such as personas and scenarios (see section 2.1).

In this chapter, we present a model-driven development approach supporting a transition from informal to formal artifacts using the MuiCSer process framework [2]. This development approach has the advantage that experts in a multidiciplinary team can continue to use the artifacts that are most familiar to them, yet allows for a smooth transition between them. Consequently, we present several tools that support different roles in different stages of the development process as one possible way of achieving support. More specifically, we offer annotated storyboards, which can be connected to other models through an underlying meta-model and cross-toolkit design support based on a abstract user interface model expressed in UIML [3]. Finally, we compare our contribution with related work and provide some further discussion.

2 Artifacts for user interface design

The design and development of interactive systems requires a deep understanding of various information sources and artifacts. Processing all information that is necessary to create a usable, appropriate and useful interactive system is often done in an engineering approach that processes this information gradually into the software and accompanying user interface. Designers need to know about the targeted end-users, the environment in which the software will be used, the devices on which it will run, the typical tasks that need to be supported and other non-functional requirements. A well-known approach that makes sure all this information does not interfere with striving for an optimal user experience, is user-centered design (UCD). Unfortunately, UCD is mostly a design approach and is hard to integrate with the actual engineering of the software. UCD is often employed by designers and delivers a user interface design, not an operational interactive system.

Connecting typical software engineering processes with UCD processes has been an ongoing challenge tackled by many researchers to date [4, 5, 6].

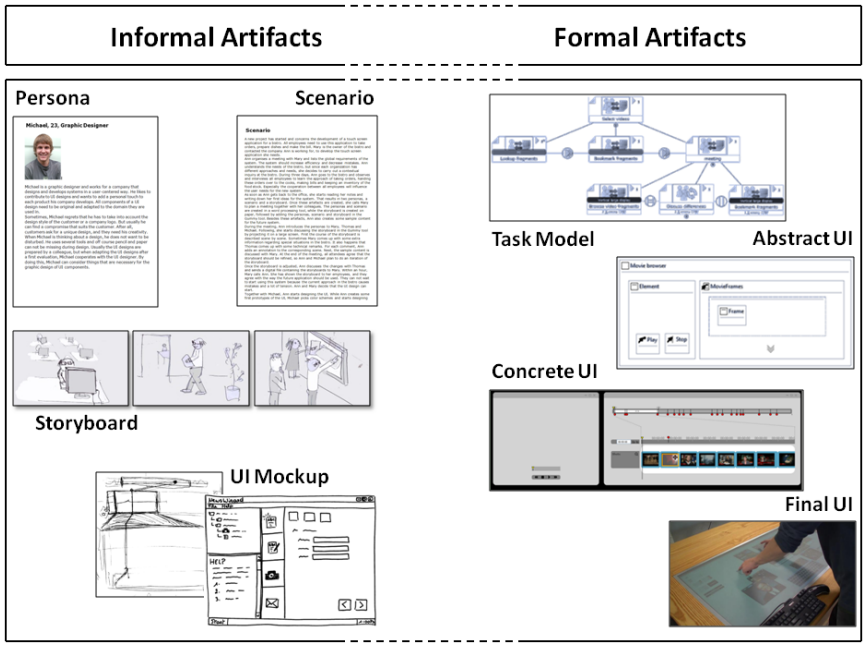

Fig. 1 Examples of informal and formal artifacts.

The gap between interface design and engineering interactive systems is mainly a matter of "design" language differences: an interface designer uses different artifacts than a software engineer. In fact, each domain of expertise uses its own vocabulary which complicates collaboration between people having different backgrounds. We now give an overview of informal and formal (i.e. adhering to a meta-model) artifacts that are used in UCD processes and by software engineers to describe user interfaces. Fig. 1 shows examples of these informal and formal artifacts.

2.1 Informal design knowledge

Informal artifacts in UCD have the advantage that they can be understood by all team members, unregarded their expertise. Informal artifacts that are often used include personas, scenario descriptions and storyboards. Natural text or an unstructured graphical representations are two types of languages often used. Typically, informal artifacts are often presented in a narrative or sketched style so as to aid a universal comprehensability.

Personas are defined as hypothetical archetypes of actual users and usually result from a user analysis [7]. A persona describes a fictional person that represents a typical group of end-users and includes personal details (such as name, age, gender

and photograph), roles, tasks, goals and skills. By personifying a group of end-users, team members are more likely to focus on the users and their needs, which may be beneficial for user experience of the resulting user interface [8].

Although personas are created in the beginning of a UCD process, they are supposed to be used during the entire process. Their primary goal is to communicate part of the user needs to all team members right after the user analysis. Furthermore, personas should be used as a guiding artifact in several other stages, such as UI design and development.

Personas contain a lot of information concerning user needs and requirements, but a similar type of informal artifacts that include user requirements are *Scenarios*. Scenarios are stories about people and their activities. A typical scenario describes a setting, includes a sequence of events or actions and represents the use of a system [9].

Both personas and scenarios have a narrative style, but in contrast to a scenario, a persona does not describe how a system is used. Consequently, personas and scenarios can be considered as complementary artifacts in UCD. In practice, personas can be included as the actors of a scenario. Both types of artifacts represent user requirements. Nevertheless, when personas or scenarios are poorly communicated or accepted by the leadership team, or when other team members do not know how to use them, a lot of information contained by these artifacts can get lost during a UCD process. Furthermore, it may be problematic for people with a technical background to translate narrative stories into technical specifications [10, 11].

The aforementioned types of informal artifacts are used as tools to communicate user requirements in a team and are presented in a narrative style. Another notation allows team members to communicate ideas visually through sketching. During many meetings or brainstorm sessions, diagrams and sketches are created or presented to express someone's ideas [12]. Sketching is very accessible, pen and paper suffice to start sketching, while even people with little drawing skills can present an insightful picture. Furthermore, sketches can be helpful to discover ideas that are very often invisible [13].

In UCD, sketches can have several shapes such as storyboards, diagrams or UI designs. *Storyboards*, originating from the film industry, can be considered as a visual representation of a scenario, and are often used to communicate ideas about a future system to stakeholders. The advantage of storyboards is that they provide a depiction of how a future system can be used that is less ambiguous than a scenario [14].

Another way to use sketches in UCD, is drawing the first *user interface mockups*. This technique is used to share ideas about what the UI should look like and to discuss several UI considerations. Furthermore, these UI mockups can be used to verify whether the first UI decisions meet the user needs. In practice, this can be done during stakeholder meetings, but also informal evaluations together with end-users can be conducted. When several screens in UI mockups are connected, they do not only present the look and feel of a UI, but also part of its behavior.

Storyboards and UI mockups do not always need to be sketched. These artifacts can also have a higher detail, depending on their aim. For instance, a storyboard or

UI mockup may be more detailed and polished when presenting it to the management or at a later stage of UCD.

Although many of the artifacts discussed in this section can be classified as infomal artifacts, part of their exact meaning is open for interpretation, they also contain unambiguous information. A notable example is the formalization of sketched interface designs using the formal specification language Z presented by Bowen and Reeves [15]. Our approach does not strive for this level of formality rather aims at structuring and extracting the information that can be linked to the models typically used in a model-driven engineering approach. The next section presents these models relevant for the design of (context-sensitive) user interfaces. This provides us with the required foundations to define a meta-model for informal artifacts such as the storyboard.

2.2 Formal artifacts

Several models are commonly used in model-driven development of context-sensitive user interfaces. The reference framework for plastic user interfaces [16] lists the different kinds of models and their role in the development process. UsiXML [17] is a single language that integrates most of these models. They list the following abstractions of the user interface, from abstract to concrete.

Task models allow to express a hierarchical decomposition of a goal, into tasks and activities, that can be translated into actions at the lowest level. The COMM task model notation [18] illustrates this by allowing specification of modalities or interaction devices in *modal tasks*, which are leaves in the task tree. They are mostly used as a first step in designing an application to identify the tasks and later actions that have to be performed to reach a certain goal.

Abstract user interfaces are high-level descriptions of user interfaces that are independent of a modality. They consist of abstract interaction objects arranged in presentation units. An *Abstract Interaction Object* (AIO) is a part of the abstract user interface that supports the execution of a leave task; it allows the user to give input to the system (e.g enter a search term) or to start the execution of some function (e.g. start the search), or allows the system to present output to the user (e.g. "searching" or the search result). Abstract user interfaces usually also define the transitions between presentation units, although the latter fact is usually not emphasized in graphical representations of the abstract user interface. Most abstract user interface languages have a formal basis in the form of a meta-model, the Canonical Abstract Prototypes notation [19] was first defined to allow more abstract paper prototyping to encourage creativity and later integrated into a modeling environment with a proper meta-model [20].

Concrete user interfaces realize abstract user interfaces for a specific context of use, such as a desktop PC used by a journalist. It already represents the final look-and-feel (e.g. following the Windows User Experience Interaction Guidelines) but is independent of the user interface toolkit (e.g. Qt, MFC or GTK). A concrete user

interface is especially useful to port user interfaces among different toolkits. Porting user interfaces between different platforms or modalities may require using a higher level of abstraction (such as the abstract user interface) to enable transition between radically different interaction objects.

Final user interfaces are instantiations of the concrete user interface for a specific toolkit (e.g. Qt on a HP PC running Windows 7). They can be interpreted or compiled to run on the target device. Final user interfaces are not considered in UsiXML or any other user interface description language supporting multiple abstractions.

Besides these abstractions there are some other models that are important to user interface design, such as the domain model, describing concepts from the domain that are relevant to the user interface, and context models that capture the context of use in terms of user, platform and environment. Although UsiXML covers a wide range of models that describe various aspects of an interactive system, it has little support for typical informal artifacts. Besides the support for low-fidelity prototypes [1], other artifacts such as storyboards and personas are not covered by UsiXML. We think a model-driven engineering approach for interactive systems needs to include other informal artifacts in the overall development process in order to support a complete design and development process.

Besides finding direct links between both informal and formal artifacts as to capture and reuse the information from the informal artifacts further down the engineering process, the process in which these artifacts are created is equally important. In the next section we focus on a process framework that structures how artifacts, originating from UCD and software engineering, are created by a multidisciplinary team, and how these artifacts make the transition from informal and unstructured artifacts toward structured and formal artifacts that define the interactive system.

3 MuiCSer process

In user-centered software engineering (UCSE) the traditional user-centered design (UCD) approach is extended toward the practical engineering (as in creation) of the software artifacts. MuiCSer [2] [2] is a process framework for *Mu*lt*i*disciplinary user-*C*entered *S*oftware engine*er*ing that explicitly focuses on the end-user needs during the entire software engineering (SE) cycle. MuiCSer embodies UCD with a structured SE approach and organizes the creation of interactive software systems by a multidisciplinary team. As such, it provides a way to define a process (based on the MuiCSer process framework) that describes when and how the transitions from informal to formal artifacts take place. The framework can be used to define a UCSE process and supports different formal models, where each model describes a specific aspect of an interactive system and represents the viewpoint of one or more specific roles in the multidisciplinary team.

[2] pronounced as “mixer”

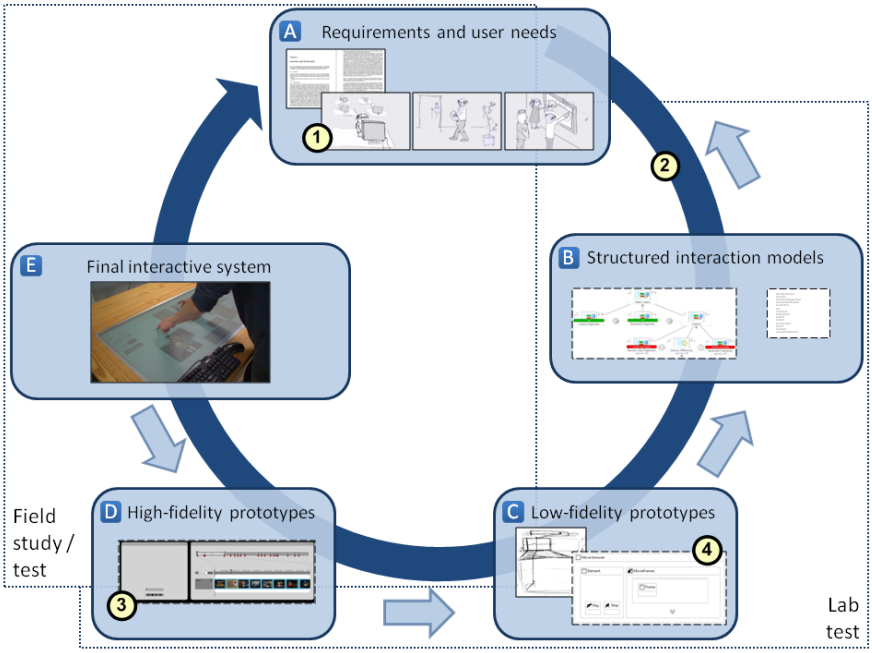

Fig. 2 The MuiCSer process that was used for the design and development of *News Video Explorer*. Extracts of the most important artifacts are presented for each stage of MuiCSer. In this diagram, informal artifacts have a solid border, while formal artifacts are distinguished by a dashed border.

Processes derived from MuiCSer typically start with a *requirements and user needs* analysis stage (Fig. 2-A) where the user tasks, goals and the related objects or resources that are important to perform these tasks are specified. Several notations are used to express the results of the analysis stage, but most of the results concern informal artifacts such as personas, scenarios and storyboards. Other artifacts that commonly result from this stage are use cases, which are more focused than scenarios but provide less context [21].

During the *structured interaction analysis* (Fig. 2-B), the results of the first analysis are used to proceed towards system interaction models, and a presentation model. These formal models are often expressed using the UML notation, thus keeping in pace with the traditional SE models. Both the informal and formal artifacts created so far, are used by user interface designers to create UI mockups during the *low-fidelity prototyping* stage (Fig. 2-C). In subsequent stages, low-fidelity prototypes are transformed into *high-fidelity prototypes* (Fig. 2-D), which on their turn evolve into the *final interactive system* (Fig. 2-E). For this evolution, informal artifacts as well as formal models are used.

Fig. 2 shows an overview of the stages of a MuiCSer process for the design and development of *News Video Explorer*, an application for TV researchers to browse a vast video archive in order to search suitable video fragments. We use this example

throughout the remainder of this chapter to illustrate our approach. At the beginning of the process (stage *A*), requirements and user needs were captured by conducting user observations that involved several professional TV researchers. The results of this analysis were a scenario and a storyboard that among others defined that News Video Explorer needs to be available in several situations: it should run on three different platforms, namely desktop pc, large multitouch display and mobile device. Based on these informal artifacts, structured interaction models (stage *B*), including a task model and a context model, were created. These models, as well as the other artifacts were used as input for sketched UI mockups and Canonical Abstract Prototypes (stage *C*), which were later translated into more detailed designs (stage *D*) and a final interactive system (stage *E*).

A MuiCSer process increases the traceability and visibility during the process. By combining UCD and SE in one integrated process, informal artifacts such as a list of requirements and low-fidelity prototypes can be made an explicit part of the process and their influence on other artifacts can be traced. By evaluating the result of each stage in the process, the support for user needs and goals and the presence of required functionality is verified.

Evaluation of informal and formal artifacts often differs: while team members have the background to read and understand the formal artifacts, informal artifacts can also be evaluated by end-users. Thus, the latter are suitable for verifying requirements with potential users and clients while the former are required to create the concrete system implementing these requirements.

A survey that involved practitioners of companies active in UCD, showed that there is insufficient support to translate artifacts created by non-technical team members into a notation appropriate for software engineers or developers [22]. Although the overall process to improve this situation is tackled by the MuiCSer framework, the different viewpoints of a multidisciplinary team and notations in UCSE still cause difficulties in the transition between some types of artifacts. From a study on the alignment of tools and models for multidisciplinary teams, we conclude that it is difficult to communicate informal artifacts on design decisions between designers and developers with existing tools [2]. There is a clear need for a well-defined approach linking informal and formal artifacts without requiring team members to get fully accustomed with each others' roles. In the next section we present a model-driven approach to accomplish an elegant transformation between informal and formal artifacts without interrupting the typical team member activities.

4 From informal to formal artifacts with storyboards

One of the biggest challenges in integrating informal design knowledge and formal models is introducing a common language that enables multidisciplinary teams to collaborate in creating advanced user interfaces. We found inspiration in the intersection of storyboarding, comics and model-driven engineering. In this section, we describe a storyboarding approach and tools that take into account storyboards

for the creation of more formal artifacts in MuiCSer processes. The proposed tools are the COMuICSer tool for creating storyboards (Fig. 2-1), mapping and transformation support for UsiXML (Fig. 2-2) and the Jelly high-fidelity prototyping tool (Fig. 2-3).

4.1 Storyboards: graphical narrative models

COMuICSer [3] [23] is a tool we provide to support the integration of informal and formal models through usage of a common visual language. It supports the MuiCSer process framework and uses a natural, visual language. This is an effective and clear way for communication amongst a multidisciplinary team, given that no other universal languages for doing requirements engineering with multiple disciplines involved exist. The graphical notation, typical for storyboarding, makes complex details comprehensible and even allows to add contextual data.

In COMuICSer, a storyboard is defined as: *a sequence of pictures of real life situations, depicting users carrying out several activities by using devices in a certain context, presented in a narrative format*. This specific definition immediately provides us with a clear overview of the four primary pieces of information that can be found in a storyboard: users, activities, devices and context.

The accompanying COMuICSer tool is shown in Fig. 3. This tool is mainly used between the requirements and user needs specification and the creation of structured interaction models in the MuiCSer process, shown in Fig. 2-1.

The COMuICSer tool allows members of a multidisciplinary team to load a scenario (Fig. 3-1). When creating scenes (Fig. 3-2), each scene in the storyboard can be connected to a fragment of the scenario, and hence a connection between the storyboard and the scenario is provided.

By annotating scenes, it is possible to provide a connection to structured engineering models and UI designs. In the COMuICSer tool, rectangles can be drawn on top of scenes, to specify particular annotations (Fig. 3-2). These annotations are made in a similar way as the *photo tagging* features on Facebook or Flickr and can concern personas, devices, activities and free annotations. The tool provides forms to specify each annotation (Fig. 3-3).

Fig. 3-4 shows an enlarged scene of the storyboard that describes the use of News Video Explorer, introduced in section 3. In this scene, we can identify three personas: the TV researchers; one device: a large multitouch display; and two equivalent activities: browsing the videos in News Video Explorer. Besides these annotations, the scene shows more contextual details such as the connections between personas, device and activities and the fact that the scene takes place in a room, where people are standing in front of the device.

These annotations and the scenes themselves implicitly capture this information in a model that conforms to the storyboard meta-model. This meta-model forms the

[3] pronounced as “comics-er”

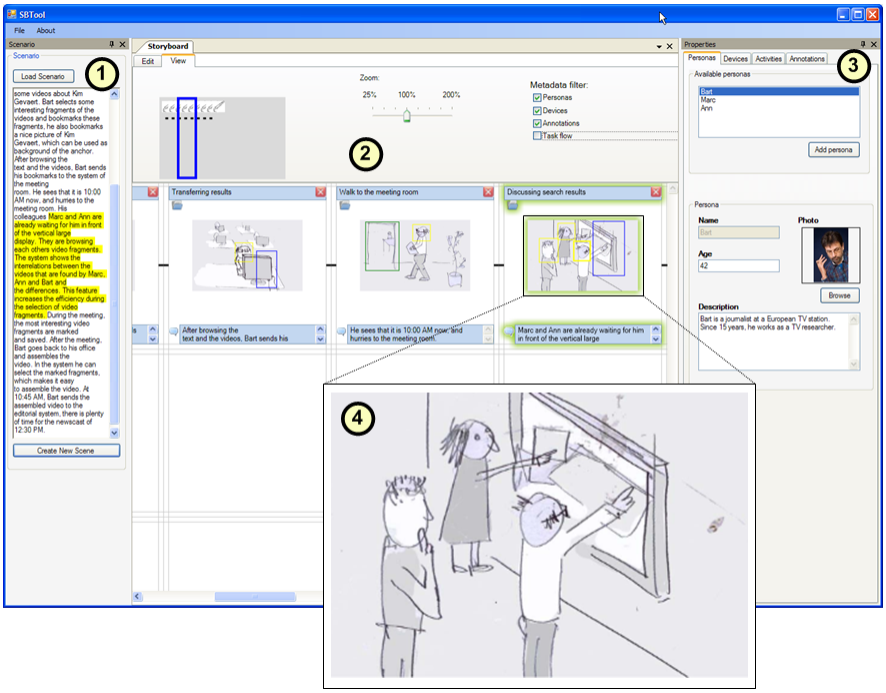

Fig. 3 Screenshot of the tool for COMuICSer. In the tool, it is possible to load a scenario (1), create scenes in the storyboard panel (2) and annotate scenes (3). One scene is enlarged (4) to exemplify what annotations and contextual information can be available.

basis for integration of COMuICSer storyboards with other models. This avoids a completely manual transformation of high-level requirements that are contained in a storyboard but at the same time does not exclude the creative input that is often part of the storyboarding process.

4.1.1 A storyboard meta-model

Our storyboard meta-model, shown in Fig. 4, is MOF-compliant[4]. There is one central element in the meta-model: the *Scene*. A scene is a graphical representation of a part of the scenario. A set of scenes that are related using *TemporalRelationShip* elements in a *Storyboard*. The TemporalRelationShip element is based on Allen's interval algebra [24]. The *before* relationship indicates one scene happened before another, and there is undefined time progress in between scenes. The *meets* relationship indicates one scene is immediately followed by another scene, and the time progress between two scenes is virtually none. Although the most common relationships used in storyboarding are *before* and *meets*, we think parallel activities

[4] MOF is an industry standard established by the Object Management Group

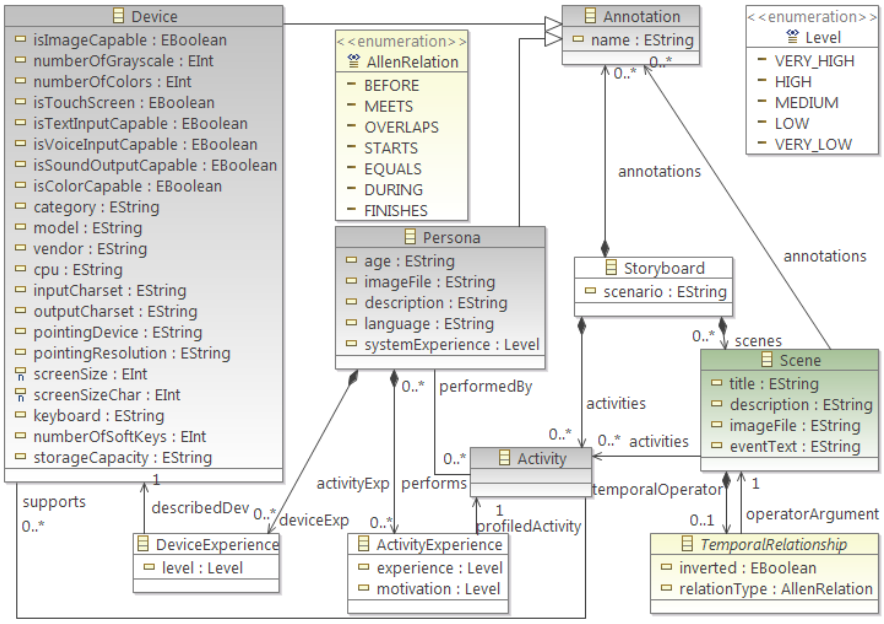

Fig. 4 Our initial COMuICSer storyboard meta-model. It contains the graphical depiction with the objects of interest (context), personas, devices and activities. Scenes are related using the Allen interval algebra operators.

should be supported since they are common in collaborative and multi-user activities. Defining more precise temporal relationships between scenes, allows us to exploit them later on, e.g. by mapping them on the temporal relationships that are used in the task model.

When constructing a storyboard, the drawings or photographs used, often contain a lot of contextual information. Dow et al. show storyboarding, especially contextual storytelling, is useful for context-aware application design (in their case ubicomp applications) but lacks a good way of formalizing the context data [10]. In COMuICSer, a scene is annotated with different types of information: *Persona* specifies archetypical users, *Device* presents computing devices and systems, *Activity* represents what happens in a scene. By providing tagging of scenes, we support a rudimentary way of translating the context inferred from the graphical depiction of a scene into a readable format. We showed a graphical depiction could have high value to obtain a usable model of the context of use in previous work [25].

4.2 Mapping storyboards to models

In this section, we show a mapping from COMuICSer storyboards to UsiXml [17] as one instance of a transition from informal to formal artifacts. We selected UsiXml as the target model because it contains a consistent set of the models used in UCSE (e.g. task model, abstract user interface model, context model, ...) and because of the available tool support[5].

The two UsiXml models that can be partially generated from a COMuICSer storyboard are: the *task model*, describing task sequences required to reach the user's goal, and the *context model*, specifying the application's context of use.

Tasks of the UsiXml task model are in direct relation with activities that are depicted in a scene of a storyboard. For each activity, the *activity2task* transformation (formalized in Fig. 5) creates a task in the task model with a *name* that corresponds to the activity *title*. Some domain-independent properties of an activity are also transformed into task properties through a one-to-one mapping. For example, the activity *importance* and *frequency* are directly mapped to their task equivalent, after a format adaptation. However, not all mappings are that straightforward: for instance the type of task can be either a system task, a user task or an interactive task. The task *type* can be inferred by the number of devices and personas related to the activity. If an activity is performed by a persona using no device, the corresponding task is a user task, if no persona is performing the activity, the task is a system task, otherwise it is an interactive task. A task is set as *cooperative* if multiple personas are involved in the performance of corresponding activities. The scene shown in Fig. 3-4, includes a cooperative task which involves two personas carrying out the same activity: browsing the videos in News Video Explorer. The same reasoning on the number of devices is used to infer if a task is a *multi-device* task or not.

```
rule activity2task{
 from a : Mstoryboard3!Activity
  to task : MUsiXmlTask!Task (name <- a.title,
    importance <- a.getUsiXmlImportance(),
    multidevice <- (a.performedUsing->size()>1),
    frequency <- a.getUsiXmlFrequency(),
    cooperative <- (a.performedBy->size()>1),
    type <- if (a.performedBy->size()=0) then 'system'
     else if (a.performedUsing->size()=0) then 'user'
       else 'interactive' endif endif)
 }
```

Fig. 5 Activity to task ATL transformation.

Two other generative transformations, the *persona2userStereotype* and the *device2platform*, focus on the partial generation of a context model from information specified in a COMuICSer storyboard. Firstly, the *persona2userStereotype* trans-

[5] The UsiXML website (http://www.usixml.org) gives an overview of the avaible tools.

formation creates a new *userStereoType* element for each *Persona* element, with direct mapping of properties such as *DeviceExperience* and *ActivityExperience*. Secondly, each *Device* is transformed to new*hardwarePlatform* and *softwarePlatform* elements. All the device properties related to hardware are mapped to their hardwarePlatform equivalent: *category*, *cpu*, *inputCharset*, *isColorCapable*, *screenSize*, *storageCapacity*, ... The rest of the device properties (*osName*, *osVersion*, *isJavaEnabled*, *jvmVersion*, ...) are mapped to softwarePlatform properties.

The mapping to the context model, straightforward as it may seem, does not retrieve all the information available in the COMuICSer StoryBoard. Indeed, the StoryBoard model provides by definition a lot more context information mainly all the entities in the environment. We generate a new context model for every scene in a storyboard, even though its likely that some of this context can be shared across scenes. Nevertheless, UsiXml does not allow to do this because variations are not possible within one context model.

4.3 Generation of model constraints from storyboards

Since storyboard operators and task operators are expressed in different paradigms, a one-to-one mapping between the two is not desirable. Indeed, COMuICSer storyboards specify the relation between scenes using Allen interval algebra... [24] with operators such as *meets*, *after* and *overlaps*. However, task models, including the UsiXML task model use LOTOS based temporal operators such as *enabling*, *choice* and *independent concurrency*. Consequently, the operators of the task models can not be generated directly. What we can do however, is provide support by restricting the choices to the ones allowed by the storyboard specification. We decided to express these restrictions using OCL constraints, in order to remain at the modelling level and exploit MOF-technology.
Below we list the transformations that take a COMuICSer storyboard instance as input and generate several OCL constraints:

- Each *after* relation between scenes requires that the tasks corresponding to the activities should have an *enabling* operator between them or between their parents.
- Each *meets* relation requires is translated into a *enabling* operator. However, no task may be performed between performance of task related to operator arguments. We achieve this by computing all possible performance paths from one task to the other (after having removed all tasks that are achieved in parallel). Unfortunately, applying this complex rule has to be done manually, currently there is no simulator available for the UsiXml task model.
- Other Allen's algebra operators: *overlaps*, *starts*, *during*, *finishes* and *equals*, are all generating constraints that impose related tasks or parents of them to have a *ConcurrencyWithInfoPassing* or a *IndependentConcurrency* relation.

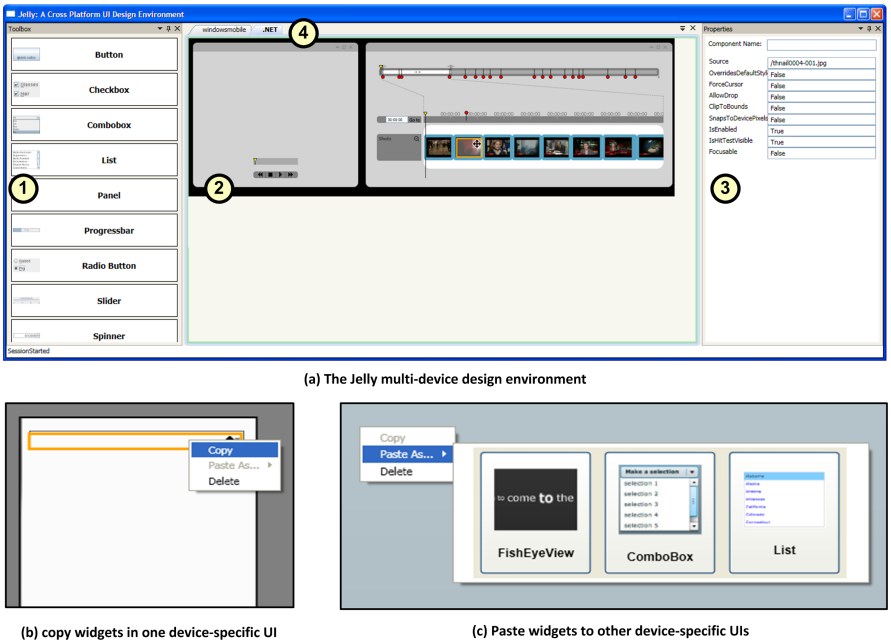

Fig. 6 The Jelly multi-device design environment, showing a video browser design for a desktop computer (a). Jelly allows designers to copy widgets from one device (b) and to paste these widgets as a similar component on a different device (c).

Finally all activities within the same scene are also valid during the same period of time. Thus, we can define that all activities of a scene are mapped on tasks from the same *enabled task set*. An enabled task set is a definition from the ConcurTaskTrees language that specifies a set of tasks is valid during the same period of time. Unfortunately, it is not possible to generate an OCL specification of this constraint, the enabled task set computation has not yet been integrated in the UsiXml tools.

4.4 From storyboard to high-Fidelity prototype

Stage D of the MuiCSer process framework (Fig. 2-D) involves the design of high-fidelity prototypes. In the MuiCSer process of the News Video Explorer, high-fidelity prototypes were created using the multi-device Jelly [26] design tool (Fig. 6 (a)). Jelly allows designers to design a concrete user interface (concrete UI), while it automatically maintains an abstract UI presentation model for this concrete design. This allows designers to use a familiar notation, in this case a graphical user interface design, and to use this artifact in a model-driven engineering process without forcing them to change their working practices considerably. The abstract UI presentation model is a structured well-defined model (based on UIML [3]).

Jelly can use storyboards that are created by the COMuICSer tool. For every device that is *tagged* in the storyboard, Jelly provides a separate design workspace. This way, Jelly automatically takes the different contexts of devices into account as indicated by a storyboard. For example, loading the storyboard for News Video Explorer (Fig. 3) results in three design workspaces: desktop pc, large multitouch display and mobile device. In each of these workspaces, designers create User Interfaces (UIs) by placing widgets from the *toolbox* (Fig. 6 (a)-A) on the *canvas* (Fig. 6 (a)-B) and dragging them around until the resulting layout is visually appealing. These widgets' properties can be changed through the *properties panel* (Fig. 6 (a)-C). This usage model is very similar to traditional GUI builders, allowing designers to reuse their knowledge of single-platform UI design tools in a multi-device design environment. Under the hood, Jelly builds an underlying presentation model which can be connected with other artifacts included in the MuiCSer process.

The mappings between Jelly's underlying presentation model and formal artifacts such as Abstract Interaction Objects (AIOs, as introduced by the reference framework in section 2.2) also help designers during the creation of high-fidelity prototypes. It introduces the flexibility to *copy* a part of a UI on one device (Fig. 6 (b)) and to *paste* it *as* a similar part on another device (Fig. 6 (c)). A component is considered as similar if it has the same *AIO type* and *content datatype* as the given component [27]. We currently support four types of AIOs, differentiated according to the functionality they offer to the user: (1) *input* components allow users to enter or manipulate content; (2) *output* components present content to the user; (3) *action* components allow users to trigger an action; and finally (4) *group* components group other components into a hierarchical structure. As content datatypes, we currently support the five primitive types of XML Schema (e.g. decimal, string, void, etc.), a number of datatypes that are often used in user interfaces (e.g. Image, Colour, etc.) and container datatypes that group content items of a certain type together (e.g. a list of strings, a tree of images, etc.).

For example, assume we are looking for all Adobe Flex components that can represent a Windows Mobile combobox. Since the combobox is linked to an input AIO, the network is first searched for all Adobe Flex components that are linked to an input AIO. This returns a huge list of controls such as checkbox, spinbox, listbox, combobox, a custom fish-eye view container, etc. Secondly, this list is searched for all components that support the same datatype as the Windows Mobile combobox (i.e. a list of strings). This finally results in three Adobe Flex components: listbox, combobox and custom fish-eye view container. These components are then displayed in Jelly's *"paste as... "* menu (Fig. 6(c)).

5 Related work

We are not the first to address the difference in "design language" between different roles in a multidisciplinary team and to bridge between informal and formal artifacts.

A study of Truong et al. [14] reports the need for a tool that offers all the functionality necessary to create storyboards. Existing storyboarding tools such as Comic Life[6], Celtx[7] and Kar2ouche[8] support the creation of storyboards, but do not take into account the characteristics of UCD processes.

Dow et al [10] present a speculative next generation storyboarding tool for ubiquitous computing. This concept of a tool provides a communication mechanism for different roles in a multidisciplinary team and supports the connection with other tools. This concept was implemented in ActivityDesigner [28], a tool that supports an activity-based prototyping process. One of the first steps supported by the tool, is the creation of a scene panel, based on everyday observations, accompanied by roles of users and activity models. These first models can contribute to the first prototypes, that can include interaction sequences. Furthermore, the tool supports the evaluation of the prototypes created. This work overlaps stages *A* (Requirements and user needs), *B* (Structured interaction models), *D* (High-fidelity prototyping) in the MuiCSer process framework (Fig. 2).

SketchiXML [1] allows creating concrete user interface models through sketches by using sketch recognition. CanonSketch [29] offers different synchronized views on a user interface presentation model (UML class diagrams with Wisdom stereotypes [30], Canonical Abstract Prototypes [19] and final user interface). This approach allows immediate switching between the different views on the user interface presentation model. Both tools mainly address step *C* (low-fidelity prototypes) and the transition to *D* (high-fidelity prototypes) of the MuiCSer process framework in Fig. 2, but in different ways. While SketchiXML mainly focuses on informal specification of models, CanonSketch focuses on fluent transitioning between working styles; from detailed design to more high-level and abstract (re-)structuring of the user interface.

Some initial work has been proposed to support transitioning from informal artifacts to an initial task model. CTTE [31] has some limited support to identify tasks in a scenario description to ease the creation of a task model based on a scenario. Although this is a useful feature, it identifies only part of the information contained in a scenario; much information contained in the scenariomodel is not important to represent directly in a task model. Other types of information, such as contextual information is easily lost in this transition. While this work also addresses the transition from step *A* to *B* of the MuiCSer process framework (Fig. 2), it starts from a purely textual scenario to extract tasks, while the task and temporal relation extraction in our approach is more gradual, by first structuring a scenario in an annotated storyboard (section 3) and then guiding the transition to a task and context model using model transformation (section 4.2).

Denim [32] allows to specify important information, such as overall screen layout and navigation in a way that is easy to grasp for many people by using sketches that can be made interactive. It however does not make an attempt to go to the next step;

---

[6] http://www.comiclife.com/

[7] http://celtx.com/

[8] http://www.immersiveeducation.com/kar2ouche/

there is no model-extraction nor code generation feature. This makes Denim a nice way to make quick prototypes and to discuss design ideas. The resulting artifact is however nearly as difficult to use in the remainder of the development cycle as a plain paper prototype.

Jelly is a design environment allowing designers to create high-fidelity prototypes for multi-device applications and to to exchange design parts across devices. This is complementary with Damask [33], a multi-device design extension for Denim [32], which is also oriented towards low-fidelity prototyping instead of designing high-fidelity UIs. Most other multi-device design approaches generate UIs automatically from a higher level model. It was shown that such approaches can be effective in some specific application domains (e.g. Supple [34] and PUC [35]). However, a major drawback of such tools is the lack of support for UI designers, which are not familiar with the models and algorithms that are used to generate the final UIs [36]. Jelly overcomes this problem by hiding the models from the designers and allowing them to work on the concrete representation of a UI.

6 Discussion

This chapter discussed the problem of integrating informal knowledge into a user-centered and model-driven process to design and develop user interfaces. This problem mainly occurs during the design and development of small-scale, context sensitive applications, which can benefit of the creativity of non-technical team members, but also need to take into account technical issues. To effectively integrate this informal design knowledge, an increased involvement of a multidisciplinary team during the entire process is recommended. This means that particular artifacts should be available to all team members in order to keep track of decisions made during the process. Nevertheless, a technique to translate informal design knowledge into more formal models is also necessary to improve communication within the team.

To accomplish this, we discussed three main tools that complement existing work: the COMuICSer tool, mapping and transformation support for formal models and Jelly. The COMuiCSer tool enables the structuring and visualisation of narrative scenarios using annotated storyboards. Starting from an informal storyboard that visualizes a narrative scenario and gradually adding more structured infomation and annotations, assists all team members to understand and agree on the requirements for a future application. Furthermore, the annotations help in formalizing and augmenting the knowledge captured by the storyboard and translating into a formal model.

The model, obtained from the information extracted from the storyboard and its annotations, supports the creation of a task and context model that conforms with the storyboard through transformations that can create an initial set of tasks with temporal operators between them and constraints.

Jelly leverages the knowledge captured by the abstract interface objects (AIOs) and their relation to concrete and final interface objects to ease creation of multi-

platform user interfaces. It does this without exposing the AIOs to designers, which stimulates designers' creativity. Furthermore, the possibility to keep an eye on a storyboard while using the Jelly tool, facilitates the possibility to keep in mind the user requirements during the creation and verification of UI designs.

In our current work we are extending these tools, but are also extending the scope of our efforts to also include low-fidelity prototypes that include navigation. To enable this we are extending the Canonical Abstract Prototypes notation [19] to also cover control and data flow. This extended notation (Fig. 2-4) will be supported by a meta-model and Eclipse-based tool support. In this way, we hope to make the abstract user interface model more accessible to designers, while encouraging creativity [19] but keeping the door open for automation.

Acknowledgements This work is supported by the FWO project Transforming human interface designs via model driven engineering (G. 0296.08) and IWT project AMASS++ (SBO-060051).

References

1. Coyette A, Schimke S, Vanderdonckt J, Vielhauer C (2007) Trainable sketch recognizer for graphical user interface design. M.C.C. Baranauskas, P.A. Palanque, J. Abascal, S.D.J. Barbosa (eds.) INTERACT (1), *Lecture Notes in Computer Science*, vol. 4662, 124–135. Springer
2. Haesen M, Coninx K, den Bergh J.V, Luyten K (2008)MuiCSer: A Process Framework for Multi-Disciplinary User-Centered Software Engineering processes. Proc. of Human-Centred Software Engineering, 150–165
3. Helms J, Abrams M (2008) Retrospective on ui description languages, based on eight years' experience with the user interface markup language (uiml). Int. J. Web Eng. Technol. **4**(2), 138–162
4. Brown J, Graham TCN, Wright TN (1998) The *Vista* environment for the coevolutionary design of user interfaces. Proc. of International Conference on Human Factors in Computing Systems, 376–383
5. Chatty S, Sire S, Vinot JL, Lecoanet P, Lemort A, Mertz C.P (2004) Revisiting visual interface programming: creating gui tools for designers and programmers. Proc. of Annual ACM Symposium on User Interface Software and Technology, 267–276. ACM
6. Redmond-Pyle D, Moore A (1995) Graphical User Interface Design and Evaluation. Prentice Hall, London
7. Cooper A (2004) The Inmates Are Running the Asylum: Why High Tech Products Drive Us Crazy and How to Restore the Sanity (2nd Edition). Pearson Education, Old Tappan
8. Pruitt J, Adlin T (2006) The Persona Lifecycle : Keeping People in Mind Throughout Product Design. Morgan Kaufmann Publishers, San Francisco, CA, USA
9. Carroll JM (2000) Making use : scenario-based design of human-computer interactions. MIT Press, Cambridge, MA
10. Dow S, Saponas T.S, Li Y, Landay J.A: External representations in ubiquitous computing design and the implications for design tools. Proc. of the Conference on Designing Interactive Systems, 241–250
11. Johansson M, Arvola M (2007) A case study of how user interface sketches, scenarios and computer prototypes structure stakeholder meetings. Proc. of BCS Conference on Human-Computer Interaction (1), 177–184. BCS
12. Buxton B (2007) Sketching User Experiences getting the design right and the right design. Norman Kaufmann Publishers, San Francisco

13. Roam D (2008) Back of the Napkin: Solving Problems and Selling Ideas with Pictures. Portfolio, New York
14. Truong KN, Hayes GR, Abowd GD: Storyboarding: an empirical determination of best practices and effective guidelines. Proc. of the Conference on Designing Interactive Systems, 12–21
15. Bowen J, Reeves S (2006) Formal refinement of informal gui design artefacts. Proc. of Australian Software Engineering Conference, 221–230. IEEE Computer Society
16. Calvary G, Coutaz J, Thevenin D, Limbourg Q, Bouillon L, Vanderdonckt J (2003) A unifying reference framework for multi-target user interfaces. Interact Comput **15**(3), 289–308
17. Limbourg Q, Vanderdonckt J, Michotte B, Bouillon L, López-Jaquero V (2004) Usixml: A language supporting multi-path development of user interfaces. Proc. of Working Conference on Engineering for Human-Computer Interaction/International Workshop on Design, Specification and Verification of Interactive Systems, *Lecture Notes in Computer Science*, vol. 3425, 200–220. Springer
18. Jourde F, Laurillau Y, Nigay L (2010) Comm notation for specifying collaborative and multimodal interactive systems. Proc. of Symposium on Engineering Interactive Computing Systems, 125–134. ACM
19. Constantine LL (2003) Canonical abstract prototypes for abstract visual and interaction. Proc. of International Workshop on Design, Specification and Verification of Interactive Systems, *Lecture Notes in Computer Science*, vol. 2844, 1–15. Springer
20. Nóbrega L, Nunes NJ, Coelho H (2006) The meta sketch editor. In: G. Calvary, C. Pribeanu, G. Santucci, J. Vanderdonckt (eds.) CADUI, 201–214. Springer
21. Harmelen MV (ed.) (2001) Object Modeling and User Interface Design. The Component Software Series. Addison-Wesley
22. Haesen M, Luyten K, Coninx K (2009) Get your requirements straight: Storyboarding revisited. Proc. of IFIP TC13 Conference on Human-Computer Interaction (2), *Lecture Notes in Computer Science*, vol. 5727, 546–549. Springer
23. Haesen M, Meskens J, Luyten K, Coninx K (2010) Draw me a storyboard: Incorporating principles and techniques of comics to ease communication and artefact creation in user-centred design. Proc. of BCS Conference on Human Computer Interaction (To appear). Dundee
24. Allen JF: Maintaining knowledge about temporal intervals. Commun. ACM **26**(11), 832–843 (1983)
25. Vanderhulst G, Luyten K, Coninx K (2009) Photo-based user interfaces: Picture it, tag it, use it. In: R. Meersman, P. Herrero, T.S. Dillon (eds.) OTM Workshops, *Lecture Notes in Computer Science*, vol. 5872, 610–615. Springer
26. Meskens J, Luyten K, Coninx K: Jelly (2010) a multi-device design environment for managing consistency across devices. In: G. Santucci (ed.) International Working Conference on Advanced Visual Interfaces, 289–296. ACM Press
27. Vermeulen J, Vandriessche Y, Clerckx T, Luyten K, Coninx K (2007) Service-interaction descriptions: Augmenting services with user interface models. In: Proc. of Engineering Interactive Systems. Springer
28. Li Y, Landay J.A: Activity-based prototyping of ubicomp applications for long-lived, everyday human activities. Proc. of International Conference on Human Factors in Computing Systems, 1303–1312
29. Campos PF, Nunes NJ: Towards useful and usable interaction design tools: Canonsketch. Interact Comput **19**(5-6), 597–613 (2007)
30. Nunes NJ (2003) Representing user-interface patterns in uml. Proc. of European Conference on Object-Oriented Information Systems, *Lecture Notes in Computer Science*, vol. 2817, 142–163. Springer
31. Mori G, Paternò F, Santoro C (2002) Ctte: Support for developing and analyzing task models for interactive system design. IEEE Trans. Software Eng. **28**(8), 797–813
32. Newman MW, Lin J, Hong JI, Landay JA (2003) Denim: an informal web site design tool inspired by observations of practice. Hum.-Comput. Interact. **18**(3), 259–324

33. Lin J, Landay JA: Employing patterns and layers for early-stage design and prototyping of cross-device user interfaces. Proc. of International Conference on Human Factors in Computing Systems, 1313–1322
34. Gajos KZ, Weld DS, Wobbrock JO (2010) Automatically generating personalized user interfaces with supple. Artif. Intell. **174**, 910–950
35. Nichols J, Chau DH, Myers BA (2007) Demonstrating the viability of automatically generated user interfaces. Proc. of International Conference on Human Factors in Computing Systems, 1283–1292. ACM
36. Myers B, Hudson SE, Pausch R (2000) Past, present, and future of user interface software tools. ACM Trans. Comput.-Hum. Interact. **7**(1), 3–28