Model-based user interface adaptation

Erik G. Nilsson, Jacqueline Floch, Svein Hallsteinsen and Erlend Stav

SINTEF ICT
Norway
{egn}/{jacf}/{svein}/{sta}@sintef.no

Abstract. Most work on model-based cross-platform user interface development is based on an assumption that the user interfaces on the different platforms should be as similar as possible. Much work on mobile user interfaces claim the opposite – that user interfaces on a mobile platform should have features not applicable on a stationary one and vise versa. Exploiting contextual information in user interfaces on mobile equipment is a prime example of this. This paper focus on this dichotomy between common development and exploiting platform specific features (or having specialized versions) on each platform. Few or none of the existing model-based languages and tools for user interface development are able to combine these two needs. These aspects are initially very difficult to combine, but in the paper we present an approach that makes this possible. First we briefly present our modelling approach, we pinpoint some of the general differences between mobile and stationary user interfaces, and we present an approach to building such self-adapting systems where the adaptation is handled by generic middleware. Our approach builds on component frameworks and variability engineering to achieve adaptable systems, and property modelling, architectural reflection and context monitoring to support dynamic self-adaptation. With this as a background we investigate how the presented modelling approach may be extended and combined with the adaptive architecture to facilitate model-based user interface adaptation. Finally, we present some more general principles for how model-based approaches may be used when developing adaptive user interfaces.

Keywords. Model-based interface design. Personalization and customization of interfaces. Patterns-based approaches. Adaptive architecture.

Introduction

The last years there has been much focus on problems connected to user interfaces on mobile equipment and adaptive user interfaces within the model-based user interface development research field [e.g. 2, 4, 13, 14, 19, 20, 24, 29, 30, 34, 35]. Although some of this work [29, 30, 34, 35] use task models to reflect the differences between how the solution is used on various platforms (and thus may be used to induce

Dagstuhl Seminar Proceedings 05181
Mobile Computing and Ambient Intelligence: The Challenge of Multimedia
http://drops.dagstuhl.de/opus/volltexte/2005/372

different solutions on different platforms), most work on model-based cross-platform user interface development is based on an assumption that the user interfaces on the different platforms should be as similar as possible [11, 23, 36, 40, 41].

Much work on mobile user interfaces [e.g. 26] claim the opposite – that user interfaces on a mobile platform should have features not applicable on a stationary one. Exploiting contextual information in user interfaces on mobile equipment is a prime example of this. Few or none of the existing model-based languages and tools for user interface development are able to handle this need. In this paper we show that it is possible to combine these two views. In the paper we address how to balance the efficiency gains in the development phase obtained by having common models across different platforms with benefits for the user by exploiting the special features on each platform, especially adaptive functionality.

We address model-based adaptation of user interfaces from two viewpoints. In one section we investigate how a comprehensive modelling approach may be extended to cover adaptive user interfaces. In another section we discuss simple model-based principles for visual adaptation. It is important to stress that there is a gradual transition between these two viewpoints, and solutions in between and solutions combining these are both relevant to use.

With the increasing mobility and pervasiveness of computing and communication technology, more and more software systems are used on or accessed by a variety of handheld networked devices used by people moving around. This introduces significant and unpredictable dynamic variation both in the user needs and in the operating environment for the provided services. For example, communication bandwidth changes dynamically in wireless communication networks and power is a scarce resource on battery powered devices when outlet power is not available. Furthermore, user interface preferences change when on the move, because light and noise conditions change, or because hands and eyes occupied elsewhere. Dynamic adaptation is required in order to retain usability, usefulness, and reliability of the application under such circumstances.

A User Interface Modelling Approach Based on Modelling Patterns and Compound User Interface Components

In this section we give a brief presentation of the modelling approach presented in [24]. Below, we will show how this modelling approach can be used to combine common, model-based development of user interfaces across platforms with significant differences with adaptive behaviour – and how the approach makes development of adaptive user interfaces easier. In this section, we focus on the motivation for the modelling approach and its main principles, concepts and features.

Most model-based languages and tools suffer from a combination of two connected characteristics: the languages offer concepts on a too low level of abstraction, and the

building blocks are too simple. The building blocks available may be on a certain level of abstraction (like a *choice element* concept that is an abstraction of radio group, drop-down list box, list box, etc.), but are still fairly basic building blocks when a user interface is to be specified. With this type of building blocks, a user interface specification is an instance hierarchy of such modelling constructs on the given abstraction level. This works well as long as the same instance hierarchy is applicable on all the platforms. If the specification is to work across platforms with a certain level of differences – e.g. with large differences in screen size – there may be a need to have different instance hierarchies on each platform.

This is often handled by dividing the specification of a given user interface in two parts, one describing the commonalities across the platforms and one describing the specialities on each platform. This division must usually be done at a quite early stage in a user interface specification [23]. Furthermore, the amount of specification code constituting the platform-specific parts tends to be more voluminous than the common part. In such a situation, it is relevant to question whether a model-based approach gives any benefit over the most relevant alternative, which is to develop the user interface on each platform from scratch [25].

The modelling approach presented in [24] is using a combination of compound components and modelling patterns [5, 38]. Compound (or composite) user interface components are used to be able to have equal or similar model instances on platforms with significant differences (including traditional GUI, Web user interfaces and user interfaces on mobile equipment). Modelling patterns are used partly to obtain the necessary level of abstraction to facilitate common models across different platforms, and partly to render it possible to define generic mapping (or transformation) rules from the patterns-based, abstract compound components to concrete user interfaces on different platforms. These mapping rules are an important part of the modelling framework. As a modelling pattern usually involves a number of objects, a user interface supporting a modelling pattern must be a *composition* of different user interface components (each being simple or composite). The transformation rules describe how the *modelling patterns* are to be realized on various platforms. This means that the transformation rules must be instantiated with the same concrete classes that the patterns are instantiated with. The modelling approach also facilitates development of user interfaces that are “richer” and more dynamic than what is possible using HTML/XML technology today [22, 25].

To utilize the potential of the modelling approach, it also includes a number of different mapping to concrete representations for each abstract compound user interface component on each platform, both based on preferences, desired user interface style, modalities, etc. Fig. 1 shows how the different main parts of the modelling approach are connected – expressed using a Unified Modelling Language (UML) class model.

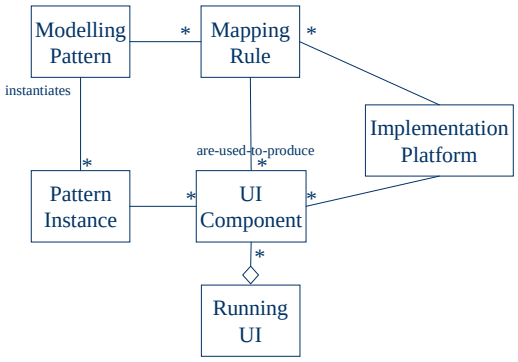

Fig. 1. Main concepts in the modeling approach

Using this modelling approach, a user interface specification consists of a number of model pattern instances, a chosen number of mapping rule instances for each of the pattern instances and additional properties specified for all of these. A specification may also include instances of patterns and/or mapping rules that are specified by the systems developer himself/herself. In addition, the modelling framework has features like extensibility (e.g. the possibility to add new building blocks and mapping rules easily) and recursive modelling (e.g. the possibility to construct new building blocks by combining existing ones).

The number of abstract components is limited – to make the modelling language comprehensible and to limit the amount of work needed to define all appropriate mappings. Yet the set is sufficiently comprehensive to render it possible to use the modelling language to specify an arbitrary user interface – it is of limited help to be able to specify and implement the wrong user interface very efficiently on a vast variety of platforms.

Mobile and Stationary User Interfaces

In this section we pinpoint some of the differences between mobile and stationary user interfaces, and argue specifically for why adaptation is of special importance for mobile user interfaces [26].

Designing user interfaces for mobile equipment is usually considered being problematic compared to designing user interfaces for stationary equipment. There are a number of reasons for this. The screen size is smaller, and the interaction mechanisms are less rich on mobile equipment; this includes both the number of available user interface components and available modalities (e.g. keyboard is not available on many mobile units). The mobile equipment is used in more demanding environments (e.g. challenging light and sound environments), and in user situations where it is difficult to use computer equipment (e.g. because the user is wearing large gloves or because the user's hands are occupied with the current task).

These problems are important and must be handled when designing mobile user interfaces. But we find it even more important to address the *opportunities* that rise from the knowledge that the user is mobile [26]. Compared to a stationary user, the context in which a mobile user operates changes much more rapidly [32]. An important challenge when designing mobile user interfaces is to exploit knowledge about these changes in the user's context – and to use this knowledge to enhance the user experience [2, 3]. The context changes are multidimensional – and sometimes rapid – and comprise position, light, sound, network connectivity, and possibly biometrics [31]. Reflecting and utilising contextual information in a mobile user interface is different and more important than in a stationary one. This causes the need for new types of user interface designs, covering the contextual aspects, and ways for the UI to adapt to changing contexts [33]. This reveals a need for something more than just down-scaled versions of desktop-UIs, which is the case for many applications and services available on PDAs and other types of mobile, general purpose computers today [12].

In many cases, exploiting the context makes it possible to simplify the user interface of a mobile user interface compared to a corresponding stationary one because much of the functionality needed to filter and/or search for data may be omitted (because it is given by the user's context). This difference may also appear when the user has to enter information. Also, the users may often benefit from a different use of media in a mobile than in a stationary application. Especially, using audio is more relevant for a mobile user in many situations – mainly because it is less intrusive for a user walking around or engaged in a task requiring visual attention on something else than a computer. For a stationary user, sound is in many cases more annoying than useful. One reason for this is that when sitting down, most users read faster than people talk, so having to wait for a person to say something that the user might as well read is often frustrating.

Adaptive Architecture

In the FAMOUS[1] project we are developing support for developing adaptive applications based on the following main ideas [7, 8]:

- component frameworks [37] as a means to build both applications and middleware that are capable of being adapted by reconfiguration;
- annotation of components and compositions with property specifications in order to aid decision making w.r.t. adaptation;
- architectural reflection [16] as a means to enable generic algorithms for making adaptations;
- implementation of context monitoring and adaptation management as generic middleware services.

[1] FAMOUS (Framework for Adaptive Mobile and Ubiquitous Services) is a strategic research programme at SINTEF funded by the Research Council of Norway

Overall architecture

The overall architecture of our approach is depicted in Fig. 2. The adaptation middleware is a component framework providing mechanisms for detecting changes in the context of applications, reasoning about these changes and adapting to them through dynamic reconfiguration of the running application. Applications are built as component frameworks from which a family of application variants can be derived by (re)configuration. The application framework is constructed to support the creation of variants matching the different requirements sets that may occur during use. The adaptation middleware is responsible for creating and maintaining a suitable configuration of the application instance based on the application framework.

The central components of the adaptation middleware are the adaptation manager, context monitor, planner and configurator. The context monitor monitors the user context and the execution environment of the application and keeps the adaptation manager informed about significant changes. When changes occur that makes the running variant of the application unsuitable in some manner, the adaptation manager will invoke the planner component, which consults the architecture model to generate plans for other possible compositions of the application. The plans are then evaluated to see how well they are suited to the current context and resource situation. If the best composition plan found is an improvement over the current composition, the adaptation manager instructs the configurator to dynamically reconfigure the application. The same mechanism is also used at application startup to find and set up the most appropriate initial variant of the application.

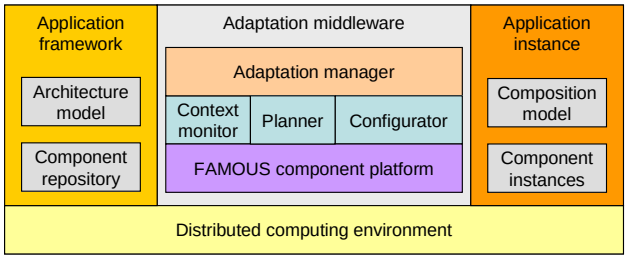

Fig. 2. Overall runtime architecture

Application variants can differ in a number of ways, for example user interface, functional richness, quality properties provided to the user, how the components are deployed on a distributed computing infrastructure, and what resources and quality properties they need from the platform and network environment. The architecture model of the application framework is a runtime representation of the application framework architecture, while the component repository stores the concrete components available for plugging into the framework.

Application framework

The application framework consists of a model of the framework architecture and a set of components fitting into the architecture. Application variants suited for different situations may be created from the framework by populating the architecture with an appropriate set of concrete components. The architecture model defines the allowable compositions.

The adaptation middleware needs a model of the framework architecture that can be represented efficiently at runtime and serve the needs of both the adaptation manager and the configurator to understand the variability built into the framework and how to configure application variants with given properties. Our approach can be seen as an example of what [6] describes as externalized adaptation based on models of the system. Our model covers the following aspects of the architecture: i) structure, ii) distribution, iii) variability and iv) property specifications. Furthermore it is intended to be derivable from design time models that are similar to models the developers are already familiar with. Our solution is based on ideas introduced by architecture definition languages [18] such as Darwin [17] and Koala [39], and adopted by UML 2.0 [27]. In addition we build on work on quality of service modelling in the context of UML [28].

Structure

The architecture model models an application as a composition of component roles collaborating through ports connected to each other (see Fig. 3). A port either defines a service implemented by the role and offered to its collaborating components, or a service needed by the role from its collaborating components in order to implement its offered services. Connections between ports are bidirectional, and the functional part of the service interaction is defined by required and provided interfaces at each end of the connection.

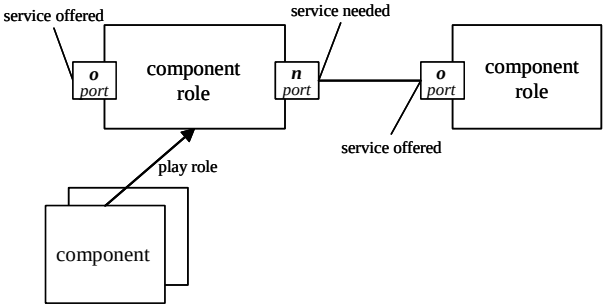

Fig. 3. Component roles and ports

Hierarchical decomposition is supported through composite components. A composite component has an inner composition of roles and associated sets of candidate components, and may be seen as a sub-framework. At the root of this hierarchical

decomposition is the role representing the application and its interaction with the user and the execution environment.

Variability and distribution
Variability is modelled by associating sets of alternative compositions to applications or composite components, or by associating sets of alternative components to component roles. Creating an application variant with given properties means selecting the appropriate alternative from each set. Distribution is modelled by associating component roles with nodes of the computing infrastructure. This association means that the component playing the role in an application or component variant must be deployed on the associated node. In this way one may vary the structure, the selected components and the distributions of the application.

Not all variability is naturally expressed in this way however. For example, a repository component on a client device may use self-defined adaptation mechanisms in order to control the degree of replication. Therefore we also allow components that manage their own adaptation. Such components must define an adaptation port that is used by the adaptation manager to coordinate the adaptation of self-adapting components with its own activities.

Property Specifications
Basically adaptation management is about matching the properties of the application to the user needs and preferences and to the properties of the execution environment. For example, if the user is driving and prefers hands-free operation, the adaptation manager should find a configuration that offers this property. In order to do so the adaptation manager needs the following information:

- the user needs and preferences (as determined by the user context);
- the properties of the execution environment;
- the properties offered by the application to the user;
- the properties needed by the application from the execution environment;
- how the properties offered by the application depend on the properties of the execution environment

To model this information, we introduce property characteristics and property constraints. Property characteristic are quantifiable characteristics of the context or of an application or component variant. A property characteristic has a name and a value range. The value range may be specified as string, integer (optionally with range indicated), enumeration (with allowable values listed) or boolean. Some examples of property characteristics are given in Table 1.

Table 1. Example property characteristics

Name	Value range	Explanation
rsp	integer	Response time
mem	integer	Amount of memory
com	integer	Bandwidth of the network connection
haf	yes, no	Hands-free operation

A property constraint limits the allowed values of a property characteristic and typically expresses a need or an offer regarding a particular property characteristic. For example, using the property characteristics listed in Table 1, the constraint "haf = yes" associated with the user, indicates that the user needs hands-free operation, while the same constraint associated with a variant of an application, indicates that this variant offers hands-free operation.

Property characteristics and property constraints are similar to quality of service characteristics and quality of service constraints as defined by the proposed UML profile for modelling QoS and Fault tolerance submitted to OMG by I-Logix, Open-IT and THALES [28]. However, property characteristics can also be used more generally to describe properties other than quality of service, like functional richness and offer or need for various types of resources.

We associate property constraints with ports to describe the properties of the service associated with the port. In the case of a service offered, the property constraints associated with the port describe the offered properties. In the case of a service needed, the property constraints describe the needed properties. The properties of a composition or component are the aggregation of the properties of its ports.

Often the offered properties of a composition depend on the properties of the services it needs or on the properties of its constituent components. This is supported by allowing a property constraint to be expressed as a function of other property constraints.

For example a property constraint describing a service offered by a component can be expressed as a function of one or more of the property constraints of the services offered to the component (and used in a given configuration), and/or of the property constraints of its constituent components.

The language for specifying such functions are simple arithmetic expressions where property constraints used as operands are referred to by characteristic name, qualified by the port name, and for properties of roles in compositions, also the role name. Some examples of property annotations are given in Fig. 4.

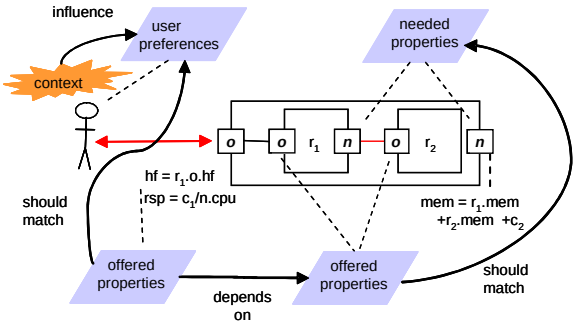

Fig. 4. Ports and property constraints

Adaptation management

The FAMOUS component platform defines a reflective component model using ports and connections, and supporting runtime reconfiguration. This foundation simplifies the tasks of the adaptation manager and the configurator in adapting the running application variant based on the best match from the architecture model.

When a context change occurs, this is detected by the context monitor which notifies the adaptation manager. The adaptation manager searches the framework architecture model for the configuration that best fits the current context and resource situation of the application. To do this it computes a utility value for the different application variants with respect to the user preferences and properties of the execution environment. Utility functions are defined by the developer; they are typically weighted means of the differences between the offered and needed property constraints. The variant with the highest utility is chosen. In order to avoid unnecessary frequent system adaptations, the Adaptation manager should also consider whether the improvement following a reconfiguration is justified.

During the configuration phase, the configurator attempts to perform only the minimum number of changes to the composition of the application. Changes can involve creating, replacing and removing component instances, relocating component instances to other nodes, and adding and removing connections between components. To make sure that these changes do not corrupt the current execution of the service, the component configurator pattern has been applied [1]. This means that affected components are requested to suspend their activity before the reconfiguration occurs and to resume it when the changes are completed.

Using Modelling Patterns and Compound User Interface Components to Facilitate Adaptive User Interfaces

The modelling approach presented in section 2 above may be used both as a means in the design and run time phases of user interface development. The description above focus on applying the modelling approach to design time needs. In this section we focus on the potential in the run time phase – where the approach may be used to realize fairly advanced adaptation mechanisms.

Normally, a model-based systems development tool does the mapping from the user interface models to concrete user interfaces in the design phase (e.g. as a code generation process), i.e. before the system is deployed to the users. As the modelling approach offers different mapping rules for each modelling pattern, our adaptation mechanism may exploit this. This causes the choice of mapping rule to use to be done in the run time phase, i.e. after the system is deployed to the users.

Using the FAMOUS adaptive architecture

As seen above, the FAMOUS adaptive architecture facilitates mechanisms for component based systems to be adapted at run time. To utilize this architecture for the presented modelling approach, a number of mapping rules must be applied at design time, so that the adaptation mechanisms have a number configuration to choose from. This may be done automatically by a code generation facility. A user interface at run time will thus consist of a number of user interface components arranged in a structure that the adaptation middleware may exploit. Fig. 5 shows an instantiation of the model in Fig. 1 at run time. The structuring of the UI components is not shown in the figure.

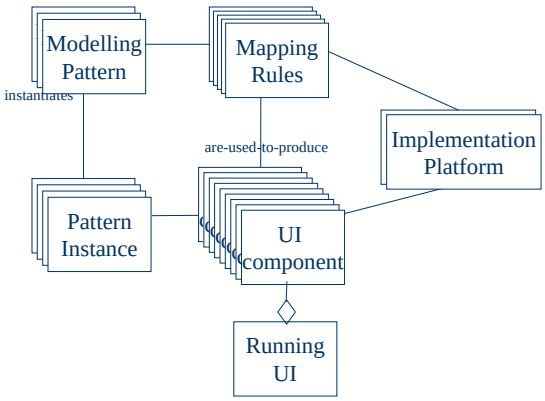

Fig. 5. User interface at runtime

What the adaptation mechanism does at a conceptual level is choosing which mapping rule to apply while an application is running. Of course, the total number of

mapping rules to apply must be decided by the developer. Whether it also will be the task of a developer to express the utility function it not obvious. At least in some cases, we envision that the utility function (or a proposition for a utility function) may be generated automatically based on knowledge of when the different mapping rules are best suited.

Types of adaptation

The most obvious types of adaptations that this approach facilitates are changes regarding the main principles for how the user interface behaves – e.g. changes in the main modality to use or change in the user interface style (e.g. from a forms based to one using icons and drag-and-drop to a wizard-based one). The reason why these types of changes are most obvious is that this is the most natural aspects to cover in different mapping rules (i.e. to have a set of mapping rules that causes the resulting user interfaces to be different from each other to a certain degree).

Although this is the most natural way of using the mapping rules mechanism in the modelling approach for adaptation at design time, the mapping rules mechanism may also be used to facilitate changes on a lower level of granularity. To make this possible, it is necessary to have mapping rules that are more similar, e.g. different versions of a mapping rule for one modality using a given style. This may e.g. be used on a mobile client to have one version that is optimal for interaction by using a stylus, and another version that is optimal for interaction by using the finger tip. The latter version would have to use larger components, and will thus have fewer interaction mechanisms on each window.

This latter use of the mapping rules mechanism to facilitate “smaller” differences in the user interface may of course cause the number of mapping rules to become larger, and if the degree of overlap between the different rules is large, changing one of them may cause the need for doing the same type of change on the parts of the mapping rule that are shared by other mapping rules. A way of handling this could be to have a sub typing mechanism that lets different mapping rules inherit from a common ancestor.

So far in this section we have focused on using different mapping rules as means for adaptation. Some low level changes may be done using the pattern instantiation mechanisms (e.g. use a different icon, table headings, sorting of lists, menus, toolbars) – this may be viewed as a different way of using the mapping rule, or as an adaptation of the mapping rule. A generalization of these issues, and other principles and mechanisms – that also are applicable to combine with the adaptation mechanisms presented in this section – are presented in the next section below.

Model-based Visual Adaptation of User Interfaces

The adaptation mechanisms and principles presented in this section are part of the adaptation features of the modelling approach presented above. It should though be mentioned that the mechanisms and principles presented in this section in addition have generic aspects that make them useable also without applying the whole modelling approach. Using the mechanisms and principles alone of course will make it necessary to implement the concrete adaptation mechanisms as part of the system that should exploit it.

While the adaptation mechanisms presented in the previous section focus on fairly large changes in the user interface (like changing modality and style to use), the adaptation principles presented in this section focus on "smaller" changes in the user interface – and how a model-based approach also facilitates this type of adaptation. By "smaller" changes we mean changes like which icon to use in a presentation, whether an icon is shown or not, which colour to use on some elements, which texts to show in different parts of the user interface, etc. Some of these types of changes (often referred to as the dialog part of an application) are usually handled by program code – and this is considered a satisfactory solution. This may also well be the case if the rules prescribing the changes are fairly simple and stable over time. If the rules are complicated, and/or they change over time, a better solution could be to use a model-based approach instead.

A model-based approach in this context is not a full-fletched modelling language with various advanced support tools, but rather a way of thinking and designing an application (i.e. more of a principle). Like the more large-scale adaptation mechanisms presented in the previous section, the adaptation functionality must be described in models that are separated from the application. But as the solutions presented in the previous section require a run time system handling the models, this is not necessary using the principles outlined in this section (but may well be used). The handling of the models may just as well be part of the functionality of the application. Which types of changes that should be applied to user interfaces when the context changes, may vary. Different contextual parameters are used to adapt *which* information to present and *how* it is presented. Instead of theorizing across a wide set of possible ways to apply these principles, we give some examples of projects where we have applied them.

In the EU project SUPREME [15], tools for supporting complicated maintenance work (e.g. in nuclear power plants) was developed. One of the areas addressed in the project was visualization of work processes. One of the prototypes developed used a simplified 2D map of a processing plant as background and icon overlays to show maintenance work and properties of the work. Figure 6 shows a part of the prototype running.

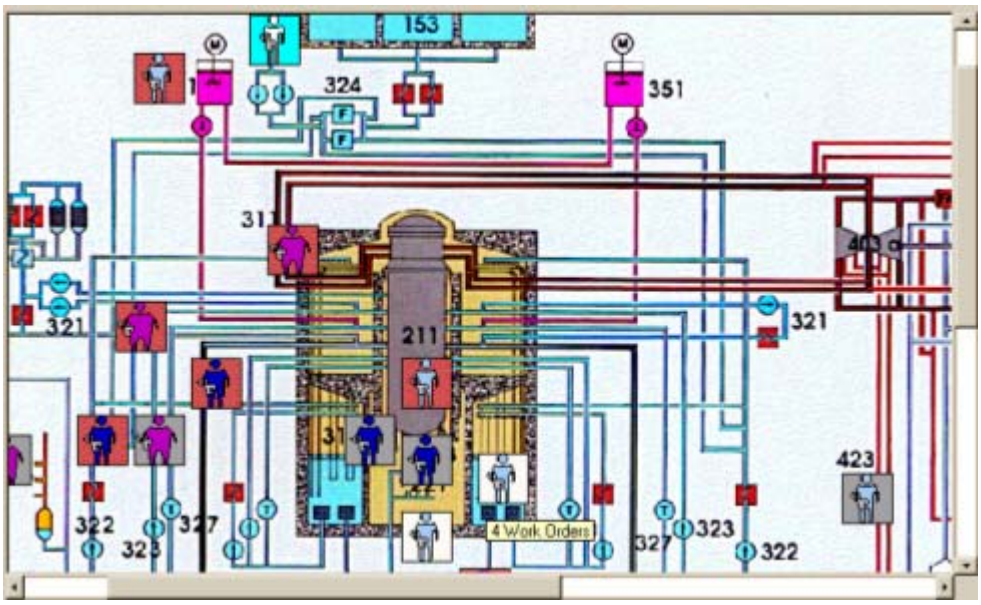

Fig. 6. Iconic 2D map visualization of maintenance work in a processing plant

The main shape of the icon tells the number of work process instances in an area of the plant (granularity of the areas is depending on the zoom level). The more work being performed, the fatter the corresponding "worker" icon. A thin "worker" icon indicates just one work order. In the prototype, this information is also redundantly coded by the foreground colour of the "worker" icons. The foreground colour could well have been used to code some other property. The background colour of the icon indicates the prime status of the work (planned, in progress, delayed, finished), and for work in progress, the filling level indicates the amount of work still to be done (the more filling, the more work to still be done). For icons representing more than one work order, these details are only indicated if all instances have the same attribute value (unless some of the processes are delayed, in which case a greyish red colour is used as background). This is an example where different attribute values are mapped to different visual aspects (position, form, colours, etc.) of icons on a visual presentation.

Another prototype developed in the SUPREME project use a 3D bar chart to visualize the time aspects of the maintenance work [9, 10]. An important feature in this tool is a filtering feature – i.e. instead of searching for information, all relevant information is shown by default, and the user applies a set of filtering mechanisms to reduce the amounts of information to show (e.g. based on time, part of the plant, status, etc.). In this tool, models are used to configure and adapt aspects like which attributes to use for filtering, and which type of user interface mechanisms to use for each filtering attribute – as well as which attribute(s) to map to the z-axis of the 3D visualization.

In the EU project GreenTrip [22], we used these principles in the VUIM (VRP User Interface Module) tool. The VUIM tool is model-based, and facilitates basic user interface functionality for Vehicle Routing applications. In the VUIM tool, both the conceptual model and a bespoke light-weight user interface model is used to tailor and adapt the user interface. E.g. which icons to use, which attributes to include, value sets for attributes, choice of which attributes to use in different dialogs, sequences of attributes in dialogs, which attribute to use as "label" in different dialogs, etc. are specified in the models, and interpreted at run time. The VUIM tool is in fact a generic user interface run time system (for VRPs), thus showing an example of a solution in between the more simple mechanisms in the SUPREME examples, and the more complicated and complete ones presented in the previous section.

In the EU project AmbieSense [21, 26], various types of technology (like context tags, agent technology, context middleware, content management support and user interface support) was developed facilitating easy development of mobile, contextual services, e.g. for travellers. In one of the demonstrators in the project, services for users at an airport were developed. Different contextual parameters were used to adapt which information to present and how it is presented. The main features of the service was adapted to whether the user is departing, arriving or in transfer. Within each of these main modes, details were adapted. E.g. information about check in counters was only shown before the user passes the security control. If the user is handicapped, different toilets were shown on the map of the airport than if the user is not. Information about tax free shopping was only shown for international travellers. In the demonstrator service for travellers in a city, adaptation like the ones used in the SUPREME plant map example is applicable. E.g. to let different aspects the icons representing restaurants change depending on properties of the restaurant and available information – like number of stars, type of restaurant, price category, whether the menu is available electronically through the service, etc.

Conclusions and future work

In this paper we have showed that when comparing mobile and stationary user interfaces, the differences between the user situation are just as important as the restrictions on screen size, interaction mechanisms etc. To design usable mobile applications, exploiting context changes is of vital importance. The rapid context changes in a mobile setting cause the need for flexible and adaptive user interfaces that are multitasking and possibly exploiting multiple modalities.

We have briefly presented a patterns-based modelling approach based on abstract, compound components and mapping rules to various target platforms. As both the number of such components (i.e. supported modelling patterns), the number of mappings for each pattern, and the number of target platforms are limited, it is possible to optimise the mappings with regard to usability and exploiting special features on each platform.

We have presented a middleware centric approach to supporting the building of applications capable of adapting to a dynamically varying context as is typical of mobile use. The proposed approach builds on the idea of achieving adaptability by building applications as component frameworks from which variants with different properties can be built dynamically.

In the paper we have shown how the modelling approach may be extended to cover adaptable user interfaces at run time exploiting the adaptation middleware. We have also discussed how a model-based approach may be used to realize adaptive features in user interfaces independently of the presented modelling approach.

At the current stage, the adaptation middleware is more mature than the modelling approach (e.g. we have implemented the adaptation middleware). Still, there are a number of challenges for both. For the adaptation middleware, making the optimization process connected to the utility function more efficient, especially for applications with many components, is very important. Also the architecture needs further development and experimentation. The modelling approach needs further refinement and details, both regarding the modelling patterns and mapping rules and how they should be used at design time, and how the mapping rules should be used to exploit the adaptation middleware to facilitate adaptive user interfaces at run time.

Acknowledgements

The work on which this paper is based is funded by the projects FAMOUS and UMBRA founded by the Norwegian Research Council, and the EU IST project AmbieSense.

References

1. Buschman, F., Meunier, R., Rohnert, H., Sommerlad, P. and Stal, M., *Pattern-oriented Software Architecture: A system of Patterns*. John Wiley & Sons, 1996. ISBN 0-4719-5869-7.
2. G. Calvary et al.: Plasticity of User Interfaces: A Revised Reference Framework. Proceedings of Tamodia'2002.
3. J. Coutaz and G. Rey: Foundation for a Theory of Contextors. In Proceedings of CADUI ‘02
4. J. Eisenstein, J. Vanderdonckt and A. Puerta: Applying Model-Based Techniques to the Development of UIs for Mobile Computers. In Proceedings of IUI’2001
5. E. Gamma, R. Helm, R. Johnson, and J. Vlissides: Design Patterns – Elements of Reusable Object-Oriented Software. Addison-Wesley, 1995
6. Garlan, D. Schmerl, B. “Model-based Adaptation for Self-Healing Systems”,. in Proc. of WOSS ’02, pp 27-32. ACM, 2002.
7. Hallsteinsen, S., Floch, J. & Stav, E.: A middleware centric approach to building self-adaptive systems. Software Engineering and Middleware 4th International Workshop, LNCS 3437 Springer Verlag Berlin Heidelberg 2005

8. Hallsteinsen, S., Stav, E & Floch, J.: Self-adaptation for "everyday" systems. Presented at WOSS 2004 (Workshop on Self-Managed Systems), Newport Beach Oct/Nov 2004.
9. H. D. Jørgensen and E. G. Nilsson: SUPREME Visualization Tool, Whitepaper, SINTEF, Oslo, Norway 1998.
10. H. D. Jørgensen: Model-driven work management services. In Proceedings of Concurrent Engineering Conference, CE 2003.
11. C. Kolski and J. Vanderdonckt: Computer-Aided Design of User Interfaces III - Proceedings of the Fourth International Conference on CADUI, 2002
12. Kuutti, K., Small interfaces – a blind spot of the academical HCI community?, In Proc. of HCI International '99
13. V. López Jaquero et al.: Model-Based Design of Adaptive User Interfaces through Connectors. In Proceedings of DSV-IS 2003
14. K. Luyten, C. Vandervelpen and K. Coninx: Migratable User Interface Descriptions in Component-Based Development. Proceedings DSV-IS 2002
15. J. Löwgren and M. V. Howard: SUPREME Visualization Concept. Recommendations and Rationale, University of Linköping, Sweden. 1996
16. Maes, P., "Concepts and experiments in computational reflection", OOPSLA'87 Proceedings.
17. Magee, J., Dulay, N., Eisenbach, S., and Kramer, J., "Specifying Distributed Software Architectures", in Proc. of the Fifth European Software Engineering Conference, 1995.
18. Medvidovic, N., and Taylor, R.N., "A classification and Comparison Framework for Software Architecture Description Languages", IEEE Transactions on Software Engineering, 2000, pp. 70-93.
19. N. Mitrovic and E. Mena: Adaptive User Interface for Mobile Devices. In Proceedings of DSV-IS 2002
20. A. Muller, P. Forbig and C. Cap: Model Based User Interface Design Using Markup Concepts. In Proceedings of DSV-IS 2001
21. H. I. Myrhaug and A. Göker: AmbieSense – interactive information channels in the surroundings of the mobile user, In Proceedings of HCI International 2003
22. E. G. Nilsson: Using application domain specific run-time systems and lightweight user interface models - a novel approach for CADUI. In Proceedings of CADUI '99
23. E. G. Nilsson: Modelling user interfaces – challenges, requirements and solutions. Proceedings of Yggdrasil 2001, Norwegian Computer Society
24. E. G. Nilsson: Combining compound conceptual user interface components with modelling patterns – a promising direction for model-based cross-platform user interface development. In Proceedings of DSV-IS 2002
25. E. G. Nilsson: User Interface Modelling and Mobile Applications – Are We Solving Real World Problems? In Proceedings of Tamodia'2002
26. E. G. Nilsson & O.-W. Rahlff: Mobile and Stationary User Interfaces – Differences and Similarities Based on Two Examples, In Proceedings of HCI International 2003
27. OMG, "UML 2.0 Superstructure Specification", Final adopted specification. August 2003.
28. OMG, "UML Profile for Modelling Quality of Service and Fault Tolerance Characteristics and Mechanisms", OMG draft adopted specification November 2003.
29. F. Paternò and C. Santoro: One Model, Many Interfaces. In Proceedings of CADUI '02
30. C. Pribeanu, Q. Limbourg and J. Vanderdonckt: Task Modelling for Context-Sensitive User Interfaces. In Proceedings of DSV-IS 2001
31. O. W. Rahlff, R. K. Rolfsen and J. Herstad: Using Personal Traces in Context Space: Towards Context Trace Technology, In Springer's Personal and Ubiquitous Computing, Special Issue on Situated Interaction and Context-Aware Computing, Vol. 5, # 1, 2001
32. O. W. Rahlff, R. K. Rolfsen, J. Herstad: The Role of Wearables in Social Navigation, in Social Navigation of Information Space, 1999

33. Schmidt, A. et.al. Sensor-based Adaptive Mobile User Interfaces, Proc. of HCI Interantional ’99
34. A. Seffah and P. Forbrig: Multiple User Interfaces: Towards a Task-Driven and Patterns-oriented Design Model. In Proceedings of DSV-IS 2002.
35. N. Souchon, Q. Limbourg and J. Vanderdonckt: Task Modelling in Multiple Contexts of Use. In Proceedings of DSV-IS 2002
36. Pedro Szekely: Retrospective and Challenges for Model-Based Interface Development. In Proceedings of CADUI ‘96
37. Szyperski, C., "Component Software: Beyond Object- Oriented Programming", Addison Wesley, 1997 (2nd ed. 2002, ISBN 0-201-74572-0).
38. H. Trætteberg: Dialog modelling with interactors and UML Statecharts - A hybrid approach. In Proceedings of DSV-IS 2003
39. van Ommering, R., van der Linden, F., Kramer, J., and Magee, J., "The Koala component model for consumer electronics software", IEEE Computer, Vol. 33, Nr. 3, March 2000, PP. 78-85.
40. J. Vanderdonckt: Current Trends in Computer-Aided Design of User Interfaces. In Proceedings of CADUI ‘96
41. J. Vanderdonckt and A. Puerta: Computer-Aided Design of User Interfaces II - Proceedings of the Third International Conference on CADUI, 1999