Abstract User Interfaces for Mobile Processes

Sonja Zaplata, Ante Vilenica, Dirk Bade, Christian P. Kunze

Distributed Systems and Information Systems,
Computer Science Department, University of Hamburg,
Vogt-Kölln-Str. 30, 22527 Hamburg, Germany

Abstract. An important focus of recent business process management systems is on the distributed, self-contained and even disconnected execution of processes involving mobile devices. Such an execution context leads to the class of *mobile processes* which are able to migrate between mobile and stationary devices in order to share functionalities and resources provided by the entire (mobile) environment. However, both the description and the execution of tasks which involve *interactions of mobile users* still require the executing device and its context to be known in advance in order to come up with a suitable user interface. Since this seems not appropriate for such decentralized and highly dynamic mobile processes, this work focuses on the integration of manual tasks on the respective ad-hoc creation of user interfaces at runtime. As an important prerequisite for that, this paper first presents an *abstract and modality-independent interaction model* to support the development and execution of *user-centric* mobile processes. Furthermore, the paper describes a prototype implementation for a corresponding system infrastructure component based on a service-oriented execution module, and, finally, shows its integration into the *DEMAC* (*Distributed Environment for Mobility-Aware Computing*) middleware.

1 Introduction

Current mobile applications and middleware platforms are evolving into a main driving factor for *pervasive systems*. Main advantages of such environments include the use of context information for increased awareness of highly dynamic vicinities and adapting to those accordingly. Since the use of mobile devices spreads increasingly, such context-aware systems also become increasingly interesting for the execution of distributed business processes (e.g. [4, 9]). In difference to traditional approaches in this field, the concept of *mobile processes* [5] focuses on the cooperation among devices in a mobile vicinity. A mobile process therefore represents a goal-oriented composition of services which can migrate to other (mobile or stationary) devices in order to share the functionality provided by these nodes. Figure 1 shows an (abstract) mobile process migrating between three devices in dependence of the discovered context. As long as the process engine of a device is able to bind local or remote services to the activities of the process instance, it is responsible for its execution. However, in cases of failures or lack of respective resources the engine has to find other devices in order to transfer the remaining process, its state and control flow to one of them. Due to the opportunity to execute the mobile process in different

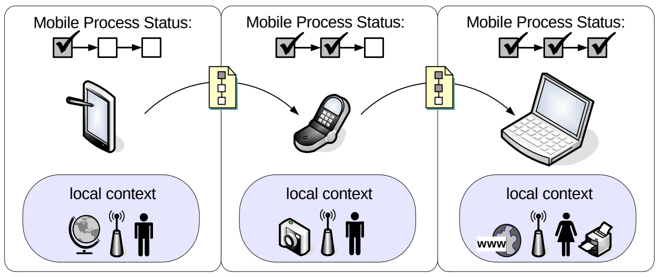


Figure 1 Context-based Cooperation: Mobile Process Execution [6]

execution contexts, this strategy avoids bottlenecks resulting from missing resources and thereby increases the probability of a successful execution [6].

Within a mobile process, activities containing (automated) applications are integrated in a single process description holding the relevant data and the current status of the process instance. The activities within such a process are identified by *abstract service classes* to refer to applications and services in a technology-independent way. This is necessary since the type of services and applications provided in the actual execution environment cannot be determined at design time − but results from the available resources and technologies during runtime. Furthermore, a mobile process can involve arbitrary tasks and heterogeneous devices − and therefore also different contexts which cannot be foreseen in advance [5,6]. Examples for such contexts are different service qualities, different network connections, and different device characteristics such as processing power, memory capacity or display size and resolution.

Depending on the type of the mobile device, also a particular set of interaction modalities is supported. Devices such as built-in car navigation systems mostly support voice output only, whereas a simple pager can only present short text messages. Other devices such as PDAs and notebooks use combined modalities (e.g. for a video-conference), and finally input-only devices such as RFID-readers are not able to present output data by themselves but have to be connected to additional hardware (e.g. a monitor). Due to this heterogeneity, there is also a great variety of possible user interfaces, so interactive tasks still have to be developed platform-dependent with a priori knowledge about the specific device and its current context.

Because of these circumstances, mobile processes are so far only realized to execute automated services. But, primary due to the fact that users carry mobile devices in order to *interact* with them, manual tasks and interactive applications play a very important role. Relevant applications for mobile processes include user notification, confirmation of activities, or the manual handling of errors occurring during process execution. A supporting middleware platform therefore also needs an abstract, modality- and platform-independent user interface model and an interaction execution module to (automatically) tailor the user interface to a specific platform at runtime. Accordingly, this paper analyses the integration of user interactions to enable the development of *user-centric* mobile processes.

The following sections of the paper summarize relevant requirements and analyse whether existing approaches can be applied to the identified problems. After a brief review of existing systems (section 3), section 4 presents the developed approach for an abstract interface model. Section 5 evaluates this approach by reporting on the development, the integration, and the scenario-based application of such user interfaces. Finally, section 6 concludes the paper with a summary and a brief outline of future work.

2 User Interactions for Mobile Processes: Requirement Analysis

The main goal of mobile processes is to support an autonomous cooperation to share mobile resources between mobile and stationary devices. As many of such processes are running in the background, an appropriate interaction module needs to emerge explicitly in case a new user interaction is required. Consequently, the interface description has to be specified within the mobile process in order to be able to migrate from one device to another − abstracting from particular interaction modalities, platforms, contexts and other device properties [5]. The descriptions must thus be *capable of being integrated into existing process description languages* and allow for *data exchange* with superordinated data flow constructs to enable control flow decisions based on data entered or automatically processed in previous (automated or manual) activities.

Since for such mobile processes all participating devices are determined at runtime, the type and the characteristics of the device which will actually execute the interactive activity cannot be predicted in advance. This means, the specific user interaction cannot be customized to the executing device, but rather depends on the resources available in the mobile vicinity. An adequate modality- and platform-independent approach for the specification of such user interfaces therefore leads to increased requirements resulting from the heterogeneity as well as from the limitations of mobile systems, e.g. restricted capacity of memory, computing power and electricity [6, 12]. Consequently, the interaction model should ideally *abstract from specific interaction modalities*, such that the most adequate one can be picked at runtime. However, in individual cases, a *specific interface modality is preferred* or even has to be fixed (e.g. viewing a picture is impossible without visual output). This means that not only the user interaction is dependent on the mobile device but also the selection of an appropriate mobile participant depends on the required interaction modality.

Finally, due to limited connectivity and relatively high costs of mobile data transfer, the process interaction model should *support disconnected execution*. This means that the execution environment must be able to process interface descriptions *without a durable connection to a central server* or to any other device. At the same time, both the description and the processing module have to be designed *considering memory capacity limitations of mobile devices*. Because of this trade-off, an approach should ideally respect both flexibility and dynamism requirements. This means, it should be possible to integrate user interactios within the mobile process in order to allow disconnected operation as well as to externalize task descriptions in order to load them at runtime, e.g. in case the mobile device has not enough capacity to store large data or it needs up-to-date information.

3 Existing Approaches

A number of research efforts have already addressed some of the requirements for realising system environments related to this work. This section briefly reviews some of these activities and analyses their respective influence, feasibility and drawbacks to be considered in an enhanced approach for mobile processes.

For the overall architecture of a mobile process management system, one of the key questions is whether to use a *thin* or a *fat client* solution. Several workflow systems exist for either of these architectures, most of them also considering user interactions. To present a user interface, thin client approaches most commonly use a locally installed internet browser to exchange information with servers in the backend using stateless HTTP or WAP protocols. Most popular technologies involve *HTML/WML*, *XForms*, *Adobe Flash* or *Java Applets* to describe visual interaction elements. Furthermore, *Ajax* (Asynchronous JavaScript and XML) is used (e.g. in *Google Gears*) to provide a richer user experience as this technology allows to partially update the user interface by asynchronously fetching new information from a server in response to interaction events. Nevertheless - because of the inherent characteristic of decentralised mobile processes - such approaches which require a centralised infrastructure have several drawbacks in general:

- Thin clients require a stable network connection to exchange information with servers, which, however, is often neither possible nor preferable (e.g. considering costs or energy consumption).
- Interaction possibilities are constrained to the elements of HTML or WML as mobile devices cannot be assumed to support browser technologies like Flash or Java applets.
- Application development is more difficult as the responsibility for managing user sessions and dealing with unreliable network connections shifts from the client to the server side.

In contrast to that, a fat client does not necessarily need to be connected to a server all the time. Instead, the whole application as well as its data is stored directly on the device, user input is processed locally and a network connection is only used to communicate results or to receive new tasks. Even fully decentralized operation is possible as data may directly be exchanged between multiple (mobile) devices. This way, applications may run independently of a network connection and sessions may be stored directly on the local device. Once a network connection is available, results may be sent back to a server or to other clients for further processing. The notion of a *rich client* takes the idea of a fat client yet another step further, as a rich client is not only designed for one dedicated task, but allows arbitrary tasks to be executed. This is often accomplished by an open, plug-in oriented and hence extensible architecture. Moreover, such architecture promotes the adaptation for different user requirements and device capabilities.

There are several corresponding fat/rich client workflow engines available. For example, the architecture presented by Pajunen [9] describes a fully service-based workflow engine, running on mobile devices. For interaction, HTML pages and forms are used and the user's input is wrapped in SOAP messages and forwarded to a local WS-BPEL workflow engine managing the control flow of the process. As another example, the *Active Forms* [10] runtime environment addresses the integration of arbitrary applications into an XHTML- or mobile forms-based user interface. User tasks are described using WS-BPEL and executed by a lightweight workflow engine to orchestrate user interactions spanning multiple applications. However, several major drawbacks can be identified for these solutions in general:

- Web-based markup languages for rendering user interfaces such as XHTML, VoiceXML, etc. constraint interaction possibilities to a specific modality and do not allow arbitrary combinations.
- Business process execution languages which require static endpoint references (e.g. WS-BPEL) are inappropriate for inter-device processes in dynamic (mobile) environments as service providers need to be known in advance.
- Common extensions for user interfaces in process execution languages (e.g. *BPEL4People* [1], or *WS-HumanTask* [2]) mostly concentrate on the specification of manual human-oriented tasks, but lack the possibility to describe a user interface in detail.

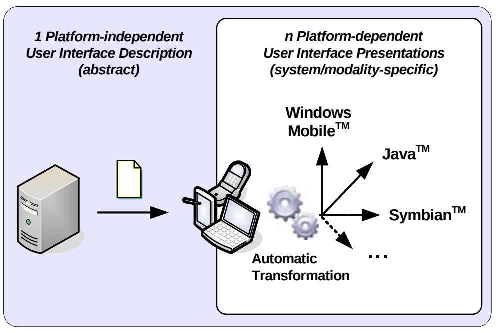
In summary, existing approaches either require a centralized infrastructure and/or lack the possibility of processing a rich description language for interactive processes. Considering languages for this purpose, approaches can be classified into *abstract* and *model-based descriptions*. The *User Interface Modelling Language* (*UIML*) [8], for example, is a representative of the abstract description family. UIML allows defining interface elements in an abstract – device independent – way, but lacks a mapping to concrete presentation components as well as the integration of control flow logic. As another example, the *Extensible User Interface Language (XUL)* [3] is not – in contrast to UIML – transformed to a specific language, but interpreted at runtime. It is used by Mozilla’s application suite to create user interfaces and is interpreted by the Gecko Rendering Engine at runtime. But as XUL can only describe graphical user interfaces, it is bound to the visual modality.

A further abstraction level is introduced by the class of *model-based interface approaches*. A popular example is the *ConcurTaskTree* (CTT) model [11] which specifies hierarchically ordered user tasks organized in a tree structure. The tree's root represents the overall task of the user in a platform-independent way, e.g. a confirmation task. The leaves of the tree structure represent the interaction components of the user interface which are necessary to fulfil this task, e.g. presenting the text to be confirmed and collecting the user's input. Although CTT thus allows designing dependencies between interface presentations, it lacks sufficient support to model data and control flow. For instance, it cannot be specified that the output of an interface should be dependent on the data values entered in a previous interface. Furthermore it is not possible to specify non-functional requirements to determine a particular behaviour of a user interface. However, CTT considers the heterogeneity of platforms and allows specifying multimodal interaction methods, which proves to be most suitable to form the basis of an abstract interface language for mobile processes. The enhanced approach based on CTT is therefore presented in the following.

4 A CCT-based Interface Model for Mobile Processes

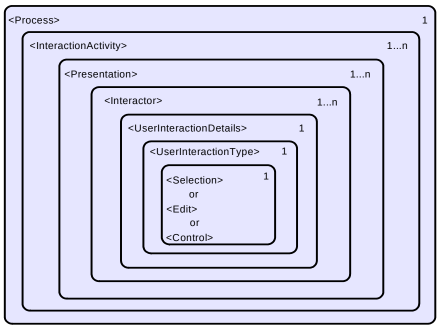
The most important function of a user interface is to present information and to capture the input of the user. Therefore, user interaction components can be characterized as (data) containers, having a set of input data which can be accessed by the user via one or more interaction modalities, and having a set of output data, which can be referenced as the result of the user interaction. The proposed abstract description language to model user interactions is therefore defined as an XML-Schema consisting of three main artifacts: One represents the description of the user's task and the respective interface specification. The other two artifacts are needed to specify the control flow and the data flow allowing the integration into existing process description languages.

In order to deal with the heterogeneity of mobile devices and to enable an efficient development of user interfaces, an instance of such a user interface description is developed only *once* and is automatically transformed into an *arbitrary number* of platform-dependent representations at runtime (cp. figure 2), taking into account the specific capabilities of each platform and also the current context of the user.



1 Platform-independent User Interface Description (abstract)
n Platform-dependent User Interface Presentations (system/modality-specific)
Windows Mobile™
Java™
Symbian™
...
Automatic Transformation

Figure 2 Schematic Diagram of the Platform-independent Model



<Process> 1
<InteractionActivity> 1...n
<Presentation> 1...n
<Interactor> 1...n
<UserInteractionDetails> 1
<UserInteractionType> 1
<Selection> or <Edit> or <Control> 1

Figure 3 Artefacts of the Abstract User Interface Model

4.1 Abstract Interaction Components for User Interfaces

Resulting from the analysis of existing approaches as presented in section 3, abstract interaction components are based on CTT. The interaction components themselves have been adopted, but have been redesigned in order to ease and enrich the platform-independent description. Furthermore, the model was enhanced to express non-functional requirements as well as stylesheets to define a specific modality-dependent representation if needed. The hierarchy of the abstract interaction components is depicted in figure 3. Herein, *Presentation* denotes a single user interface, which consists of a number of interaction components called *Interactors*. An *Interactor* represents the root element of an abstract interaction component and could for instance be mapped to a *TextField* using the visual modality or a *Prompt* using the audio modality. The concrete functionality of an *Interactor* is specified by its child elements, which are grouped by the *UserInteractionType* into the basic user interface elements *Selection*, *Edit* and *Control*. The *Selection* element is responsible for data elements that can only be selected but not edited, whereas *Edit* also allows data manipulation. The *Control* element is used to navigate within user interfaces or to activate actions, e.g. it can be used to switch between different *Presentations*. Key concept within the abstract description of interaction components is the use of descriptive attributes. These attributes characterize the intended behaviour of an element in a platform-independent way, e.g. a *Selection* can be defined as a *SingleSelection* (meaning only one item can be selected) or as a *MultipleSelection* (allowing the selection of an arbitrary number of items). Furthermore, the expected cardinality of items can be specified to enable the most appropriate representation, e.g. a *SingleSelection* with a high cardinality may be transformed to a *ChoiceGroup* realized as a *Popup* using the visual modality whereas it may be transformed to a number of dialogs using the audio modality. Furthermore, the abstract description may contain a preferred modality and a fallback, if the modality is not available on the mobile device (not depicted).

4.2 Control Flow and Data Flow Components

As mentioned above, the platform-independent model also contains artefacts to describe the control flow and data flow between several user interfaces. With this approach, the proposed model is able to support a micro as well as a macro perspective on user interactions, so called *User Interaction Processes*.

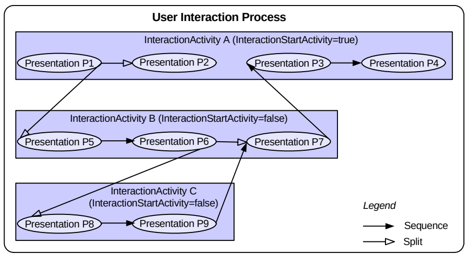


Figure 4 Micro Perspective on User Interactions

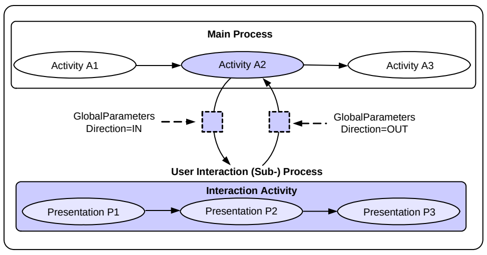


Figure 5 Macro Perspective on User Interactions

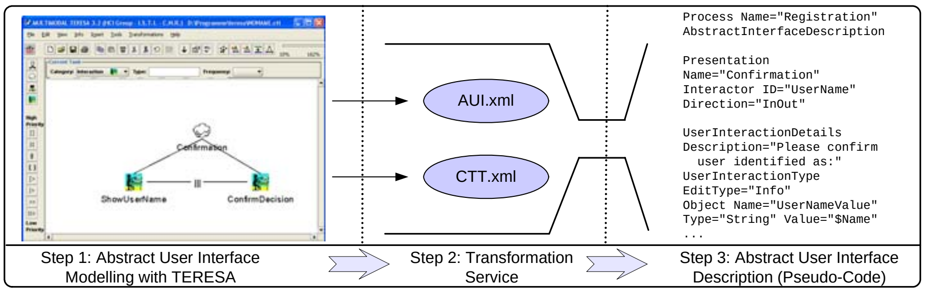
The former perspective describes a situation where the interaction process is used as a separate stand-alone service and therefore has to maintain the control flow and data flow itself. Figure 4 depicts such a stand-alone interaction service, which itself consists of a few subprocesses, called *InteractionActivities,* being connected via standard control flow elements such as *Sequence, Loop* or *Split* (cp. [13]). It is characteristic for the micro perspective that the service does not depend on other components to process the data and control flow. In contrast to this, the macro perspective characterizes a situation where control flow and data flow is (mainly) maintained by a superordinate process. Thus, the interaction process is only called when user interaction is needed. This perspective is depicted in figure 5 where *Activity A2* of the main process calls a user interaction (sub)-process. Selected parameters are passed to the subprocess and (the same or other parameters) are expected to be returned. Using the macro perspective with multiple *Presentations* requires a more powerful interaction processing module, but avoids to interrupt the interaction processing in case of several follow-up interfaces as usual for more complex interactions. Alternatively, the interaction can consist of a single *Presentation* only, so that the control flow is returned to the parent process after each interface presentation. This variant allows the realization of more simple interaction processing modules (only responsible for single interface interpretation and representation) and is therefore also suitable for less powerful devices. Being able to support these two perspectives enables developers to tailor interaction processes to specific applications by deciding which functionality should reside inside the interaction process and which functionality should be provided by the superordinated process. This flexibility therefore also allows a combination of user interactions and automated services.

5 User Interface Development, Integration and Realisation

To evaluate the developed interaction model, this section gives a brief overview of its overall applicability. Therefore, support for the development process using an existing tool is presented first, followed by the integration into the DEMAC middleware and the execution of a selected example application. Finally, experiences with the prototype realisation as well as usability aspects are discussed.

5.1 Modelling Abstract User Interfaces with TERESA

As mentioned above, the proposed model for abstract user interfaces is based on CTT, which can be graphically modelled by the toolkit TERESA [7]. In order to enable graphic modelling and to support and speed up the development of abstract user interfaces, a *transformation service* for TERESA has been developed. The proposed development process is depicted in figure 6. It starts with the interface modelling using TERESA (step 1), which generates two output files (AUI.xml and CTT.xml). These files are then inserted into the developed service (step 2) which *automatically* transforms the designed model into an enhanced abstract user interface description (step 3). Since the intermediary service performs this transformation automatically, it allows the developer to use TERESA as a graphical development tool to design abstract user interfaces for mobile processes, making it unnecessary to learn the syntax of the abstract description language.


Confirmation
ShowUserName
ConfirmDecision
AUI.xml
CTT.xml
Step 1: Abstract User Interface Modelling with TERESA
Step 2: Transformation Service
Step 3: Abstract User Interface Description (Pseudo-Code)

```
Process Name="Registration"
AbstractInterfaceDescription

Presentation
Name="Confirmation"
Interactor ID="UserName"
Direction="InOut"

UserInteractionDetails
Description="Please confirm
  user identified as:"
UserInteractionType
EditType="Info"
Object Name="UserNameValue"
Type="String" Value="$Name"
...
```

Figure 6: Development of Abstract User Interfaces

5.2 Integration with the DEMAC Process Management System

Mobile processes can be executed by a distributed execution engine such as realized in the *DEMAC* (*Distributed Environment for Mobility-Aware Computing*) project (cp. [5,6]). The respective mobile process management system uses the XPDL-based meta-description language DPDL (*DEMAC Process Description Language)* to describe the sequence of activities as well as the user's and application's non-functional demands (cp. [5,13]). Figure 7 shows the most relevant elements of the DPDL metamodel, consisting of native XPDL elements and enhancements for process migration, e.g. abovementioned non-functional requirements (modelled as *Strategies*). As shown in the figure, the abstract interface descriptions can be attached to the *workflow relevant data* without changing the structure of the process description language but inheriting existing constructs. The module which interprets the interface description is installed as an ordinary application. Thus, whenever user interaction becomes necessary, the application is called by the processes' control flow and presents the required user interface. In conjunction with the definition of non-functional requirements, this architecture also allows executing user-prioritised modalities automatically.

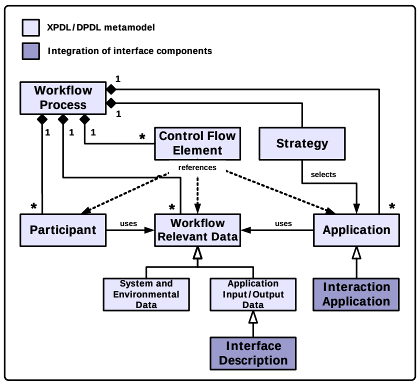


Figure 7 Integration of Interface Components into XPDL-derived Language

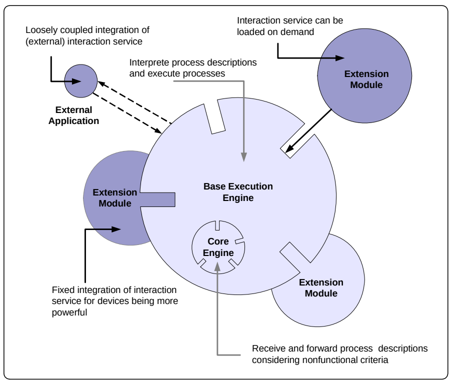

Loosely coupled integration of (external) interaction service
Interaction service can be loaded on demand
Interprete process descriptions and execute processes
External Application
Extension Module
Base Execution Engine
Extension Module
Core Engine
Extension Module
Fixed integration of interaction service for devices being more powerful
Receive and forward process descriptions considering nonfunctional criteria

Figure 8 Schematic Diagram of the DEMAC Workflow Engine (cp. [5])

To support a large range of mobile devices with different capabilities, the corresponding *mobile process execution* engine is structured modularly. A core engine module supports the migration of processes and a base execution engine is responsible for executing the control flow of the mobile process and to bind functional services to the processes' activities (cp. figure 8). The interaction module can either be installed as a fixed part of the process engine, as an extension module or as an external service component which is only activated in case a user interaction is needed, e.g. to save memory space. Due to this loose coupling strategy, the interaction module can easily be switched off, exchanged and e.g. substituted by a module for another interaction modality if applicable. However, due to its abstract structure and its independence from superordinated control flow mechanisms and process management systems, the interface description can also be applied to other process description languages such as WS-BPEL or native XPDL and their corresponding execution modules.

5.3 Example Application Implementation

As motivated in section 1, there are several application areas for user interactions within mobile processes. Most relevantly, unidirectional interactions (presenting either output or input interfaces) and bidirectional interactions (combining output and input interfaces) can be integrated and combined to realize directives, notifications or confirmations addressed to the mobile user.

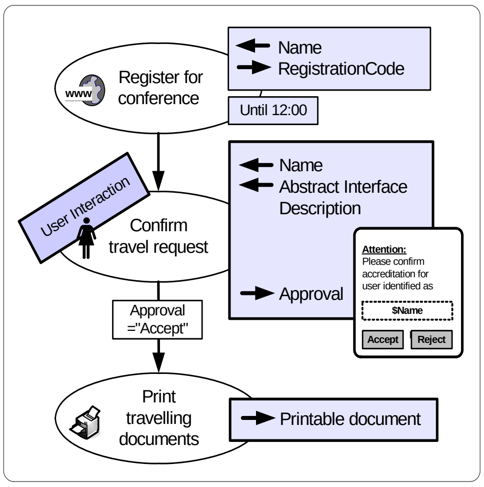


Figure 9 Example: Mobile Process with Embedded User Interaction

```
Process Name=RegistrationProcess
DataFields:
   Name (String)="John Smith"
   Approval (String)="[Accept|Reject]"
   AbstractInterfaceDescription (String)=

     "Presentation Name="Confirmation"
     Interactor ID="UserName" Direction="InOut"
     UserInteractionDetails
     Description="Please confirm accreditation
                  for user identified as:"
     UserInteractionType EditType="Info"
     Object Name="UserNameValue" Type="String"
     Value="$Name"
     Interactor ID="Decision" Direction="InOut"
     UserInteractionDetails
     UserInteractionType
     Selection  Type="SingleSelection"
     Object Name="Options" Type="String[]"
     Value="$Approval""

Activity ID="Register for conference"
  InputParameters: Name
  OutputParameters: RegistrationCode
Activity ID="Confirm travel request"
  InputParameters: Name, AbstractInterfaceDescription
  OutputParameters: Approval
Activity ID="Print travelling documents"
  OutputParameters: PrintableDocument
```

Listing 1 Pseudo-code Representation of Abstract Interface Description Example

A simple mobile process example containing a bidirectional user interaction is depicted in figure 9: A mobile researcher wants to register for a conference. As he currently does not have an internet connection to directly access the registration service and registration has to be done until a specified deadline, he initiates a mobile process to potentially use the resources of other devices in his vicinity to get his accreditation in time. Assumed that his request has been confirmed by the researcher's supervisor, the resulting travelling documents can be obtained in printable format (e.g. PDF). The graphical representation of the mobile process (figure 9) shows two service invocations (*Register for conference, Print travelling documents*) and a manual activity (*Confirm travelling request*). Attached to the activities, the required context (Internet access, special person, printer), non-functional parameters (the deadline condition) as well as the input and output data of each task are shown. As to see, the abstract interface is embedded inside the mobile processes' data container, holding the description of the elements and attributes to build an appropriate interface at runtime. Listing 1 shows a selected part of the mobile process containing the abstract interface description of the *Confirm travel request* activity as modelled with TERESA (cp. section 5.1).

For the actual execution, there are now several possibilities: Still assuming the researcher's mobile device is not able to handle the registration request, the process migrates to another (more capable) device and calls the registration service. Following, as mobile processes allow the specification of concrete participants (persons, devices or generic roles), the required person to execute the upcoming task is selected. The mobile process could therefore either migrate to the supervisor's device and present the confirmation task locally or - if applicable - invoke the respective interaction module from remote. Using the abstract description of the user interface, the interaction module decides about the modality in dependence of the current context. If, e.g. the supervisor is currently within a meeting, the confirmation is presented as a textual dialog. However, if she is driving a vehicle, audio output and automatic speech recognition are applied. Furthermore, if the interaction activity requires a particular interface modality or has to consider other non-functional aspects (e.g. display resolution) the DEMAC core engine searches for an adequate device to invoke this task, e.g. preferring the supervisor's notebook instead of her mobile phone.

5.4 Prototypical Implementation and Usability Evaluation

The interaction module receives and interprets the abstract interface description and transforms it into a device-specific representation. Except for its standard interfaces for integrating the process management system it can be realized as a device-specific component involving arbitrary sub-modules, e.g. to provide different interaction modalities. The prototypical implementation used in the DEMAC project is developed for conventional PDAs (using J2ME's CDC Personal Profile and visual modality only) and for modern sub-notebooks (using Java Standard Edition with voice output and automatic speech recognition). Without support for mobile processes, the interaction module can additionally be used as a standalone application for mobile phones, using the J2ME CLDC MIDP 2.0 profile (cp. [12]).

The respective mapping decision as well as the transformation process is transparent to the user. However, resulting from the scarcity of resources in mobile environments, the usability of the automatically generated interfaces mostly depends on the complexity of the desired artefacts and the capabilities of the platform actually used. Simple output interactions (e.g. directives or notifications) can be presented in a very consistent way over both modalities by presenting either a textual info screen or by using synthesised voice reading the message to be presented. Nevertheless, more complicated interface descriptions which require the user's input are sometimes affected by a vendor-specific interpretation of concrete interaction objects. Therefore, at least visual user interfaces differ slightly from device to device. With respect to the layout of output components this is not a usability problem, whereas the vendor-specific mapping of navigation keys and menu placements in some cases hinders an intuitive navigation (e.g. using left and right buttons for back- and forward navigation). Considering the audio modality, usability is strongly influenced by the quality of speech recognition software and hardware (e.g. microphones). For instance, within the realised example bad recordings or insufficient interpretations of spoken words sometimes lead to wrong decisions in the confirmation activity. However, the fall-back to the (more unambiguous) visual modality helps to deal with such problems in a reliable way.

6 Conclusion and Future Work

This paper introduces an abstract model to describe *modality-independent user interfaces* for mobile processes, enabling user interactions even for distributed tasks which involve several heterogeneous devices. Depending on the actual context and on the characteristics of the respective mobile device detected at runtime, the most appropriate way of interaction can thus be chosen dynamically. Moreover, the possibility to describe human tasks in a very abstract way relieves process designers from considering the system-specific behaviour of each executing device and its possible context.

In order to avoid problems caused by the cooperation of multiple heterogeneous devices using different modalities, future work will address the processing of data resulting from a user's input. A platform-independent representation of a user's input data may be necessary to ensure this information can be reused as output data in follow-up activities on devices with different modalities. In addition, use cases containing cross-modality interactions shall be analysed to further evaluate and advance the approach.

References

[1] Agrawal et al., BPEL4People Specification 1.0, Active Endpoints, Adobe Systems, BEA Systems, IBM, Oracle, SAP, 2007.

[2] Agrawal et al.: WS-HumanTask Specification 1.0, Active Endpoints, Adobe Systems, BEA Systems, IBM, Oracle, SAP, 2007.

[3] Goodger, Hickson, Hyatt, Waterson: XML User Interface Language (XUL) 1.0, Specification, http://www.mozilla.org/projects/xul/xul.html, Mozilla Foundation, 2007.

[4] Hackmann, Haitjema, Gill, Roman: Sliver: A BPEL Workflow Process Execution Engine for Mobile Devices, in: Proceedings of 4th International Conference on Service Oriented Computing (ICSOC), pages 503-508, Springer Verlag, 2006.

[5] Kunze, Zaplata, Lamersdorf: Mobile Processes: Enhancing Cooperation in Distributed Mobile Environments, in: Journal of Computers, 2(1):1-11, 2007

[6] Kunze, Zaplata, Turjalei, Lamersdorf: Enabling Context-based Cooperation: A Generic Context Model and Management System, in: Business Information Systems (BIS), 2008.

[7] Mori, Paterno, Santoro: Design and Development of Multidevice User Interfaces through Multiple Logical Descriptions. In: IEEE Transactions on Software Engineering 30(8): 507–520, 2004.

[8] OASIS: User Interface Modelling Language (UIML), Specification, Organization for the Advancement of Structured Information Standards, 2007.

[9] Pajunen, Chande: Developing Workflow Engine for Mobile Devices, in: Proceedings of the 11th Enterprise Distributed Object Computing Conference (EDOC), 2007.

[10] Pajunen, Chande: ActiveForms: A Runtime for Mobile Application Forms, in: Proceedings of the International Conference on the Management of Mobile Business, page 9, IEEE Computer Society, 2007.

[11] Paterno: Model-based Design and Evaluation of Interactive Applications, Springer-Verlag, 2007

[12] Satyanarayanan: Fundamental Challenges in Mobile Computing, in: Proceedings of the 15th ACM Symposium on Principles of Distributed Computing, 1996.

[13] WfMC: XML Process Definition Language, Version 2.0. Specification, Workflow Management Coalition, 2005.