Mobile Process Description and Execution

Christian P. Kunze, Sonja Zaplata, and Winfried Lamersdorf

Distributed Systems and Information Systems
Computer Science Department, University of Hamburg
Vogt-Kölln-Str. 30, 22527 Hamburg, Germany
[kunze|zaplata|lamersdorf]@informatik.uni-hamburg.de

Abstract. Mobile devices are increasingly aware of their respective locations and vicinity and tend to communicate rather loosely with each other; therefore *asynchronous* communication paradigms are used predominately so far for corresponding mobile applications. However, while such communication mechanisms are suitable for simple activities, they may become insufficient for more complex tasks which consist of longer sequences of related activities tied together in *application-oriented processes*. This is of particular importance if the resulting operating sequence spans several mobile devices in frequently changing vicinities.
Therefore, the work presented here provides a concept for integrating explicit support for such mobile processes into mobile system infrastructures and for distributing their execution over different nodes in the network. For this purpose, a corresponding middleware platform (extension) for context-aware mobile applications is proposed. It supports such migrating processes and helps to execute them under the restrictions typically imposed by realistic mobile applications. In particular, this paper proposes a corresponding *process description language* and an *execution model* for mobile and distributed (business) processes in the context of the project DEMAC (Distributed Environment for Mobility-Aware Computing).

1 Introduction

Due to the constraints of mobile computing environments, mobile systems, in general, cannot provide the same degree of distribution transparency as systems in statically wired environments [4]. Just in contrast to those, the *restrictions of resources* in comparison to static devices, the *increased variability in performance and reliability* of wireless connections, the *finite energy sources* to rely on, and the *hazard of mobility* itself [13] lead to the perception that mobile environments should be aware of the changing vicinity and also should react and adapt to it accordingly.

However, in current systems this so-called context *awareness* and *adaptability* is, in most cases, still restricted to support more or less monolithic and ad-hoc static applications in fulfilling their momentary tasks. In general, that means that most existing middleware systems are rather application centric and thus restricted to offer assistance for basic but rather simple tasks. But, in order to

approach the vision of pervasive computing [16, 17] more closely, also much more complex and eventually even unknown tasks and thus more generality must be supported by new mobile middleware systems.

Such complex application tasks can be regarded as sequences of related simple tasks tied together in a (business) process which is managed by a mobile client on behalf of a user. This means that a mobile client is required to reach and invoke all the services needed to execute such a process. It must also be capable of handling all intermediate results – regardless of their size and relevance to the expected final output. As a consequence, it may become a single point of failure and also a bottleneck during execution time. Altogether, this means that the capabilities of a mobile client limit the quantity of possible processes to be executed.

But since the user is, in most cases, just interested in some specific effects of a process (and not in its execution or intermediate results), this effect could be eased by transferring the control flow – and with it the whole process – to other devices, if possible. In combination with the possibilities of mobile computing middleware systems to utilise context information and to cooperate, such long-time mobile processes and their distributed execution provide additional efficiency to application process execution in mobile computing. Accordingly, this paper presents an outline of the system platform *Distributed Environment for Mobility-Aware Computing* (DEMAC) – which realises such an extension – with a special focus on a new description language and execution model for such *mobile processes*.

The following subsections of the paper introduce the definition of mobile processes, section 2 addresses related work, and section 3 provides a closer look at the coarse system architecture, the process definition language, and the execution engine. Finally, section 4 concludes the paper with a summary and an outline of future work.

1.1 Integrating Processes into Mobile Computing Systems

The work presented aims to extend the capabilities of mobile devices through cooperation with other devices in their vicinity and thus increase of their potential. This is achieved by integrating distributed (business) processes into an adequate mobile system infrastructure. Such an approach is different to most existing ones of integrating processes with mobile computing devices which just extend their traditional process infrastructure by including mobile device as process participants (cp. e.g. [12]). Accordingly, in our context, the term mobile process is defined and used as followed:

> *A mobile process is a sequence of (remote) services which may last over a longer period of time and span several devices during its execution. The results of the process are the effects the initiator expects from it.*

In traditional mobile middleware, a process executes the application logic by explicitly assigning local or remote services to the processs activities and by

invoking them directly. In contrast to that, in our view, such application processes may (partly) diffuse into the mobile middleware: They just form a stub which collects information from the user to assemble the process and its general conditions and to pass the mobile process to the middleware.

In addition, as activities of mobile processes can last very long (like hours, days, or weeks) the changes of the device environment can be dramatic between the executions of adjacent activities. Therefore, a *late binding strategy* to assign services is – certainly – essential but not always sufficient. Consequently, the mobile processes as proposed here are executed based on an *opportunistic strategy*: As long as the process engine of a device is able to bind local or remote services to it's currently activity, it is responsible for the mobile process. However, in cases of failures or lack of respective service instances the engine is able to try to find other devices which are able to execute the mobile process and then transfers the remaining process and its execution to one of them.

Such a process distribution is especially advantageous in (realistic) heterogeneous and frequently changing mobile environments where device capabilities may highly differ. Thus, such process transfer opens up additional services which were not accessible according to the traditional execution approach. This also means that likelihood of a mobile process to be executed successfully increases substantially.

1.2 Requirements for Descriptions of Mobile Processes

In order to describe processes in ways which allow for execution strategies as described above, an abstract process description language has to be designed: In such a view, mobile processes have rather similar requirements for their description as traditional (business) processes, these are among others: the need for the ability to express the *business logic* with its data and control flow, the *participating parties* (as roles or individuals), and routines to recover from *failures* [9].

But they also have some specific requirements based on the nature of mobile environments and the opportunistic and distributed execution strategy (cp. section 1.1): E.g. mobile process descriptions must be lean and simple to process in order to save memory, CPU power, and energy resources, it must also include mechanisms to handle communication failures and the distribution of the process itself. This means especially that the state of the process and the user's non-functional conditions for the execution of the process must be expressible. The (late) binding mechanism to assign service instances to process activities as late as possible must be integrated into the description language by using a preferably very abstract notation of the desired services [13, 7].

Based on these "related work" is briefly reviewed in section 2 and for mobile process description languages in section 3.2.

2 Related Work

Since this paper concentrates specifically on the description and execution of mobile processes, some specific aspects of our approach are pointed out first - before, after that, related work in the area of mobile process descriptions is reviewed more extensively.

System Infrastructure Since mobile process execution always relies on contextual information, the context modelling and context data acquisition are crucial for the respective developed concept and system infrastructure. The abstract and generic definition of context and its data as used in the *Context Toolkit* [5] by Dey is mainly suited for the mostly a priori unknown demands of mobile processes. Whereas the understanding provided by Schilit [14] or Schmidt [15] turned out to be too narrow to support the wide range of possible processes as required in our approach. The idea of the *NEXUS* project [6] to ensemble the context of an entity by federating local context clippings of entities within particular vicinity is used in the system infrastructure to construct a global context representation efficiently.

The mobile process infrastructure as addressed here also relates to recent research in the area of *mobile agents* [3]. However, in relation to that it differs in some important aspects: In contrast to an agent a mobile process does not contain executable code. In fact, mobile processes only provide meta-data about the structure of the described application and, thus, the estimated effects but not the way how this behaviour is achieved. In addition, they do not have a *social behaviour* either, nor could they act *autonomously* or *proactively*. Nevertheless, some parts, e.g. security and privacy concerns or the need to determine the execution state, have, in principal, similar requirements and, thus, solutions.

Process Description A process description language for mobile processes has to consider aspects of distribution as well as support for high level flexibility and fault tolerance. An analysis of most prominent existing process description languages, such as *XPDL*, *BPEL4WS*, *WSCI*, *JPDL*, and *ebBPSS*, shows that the concepts and constructs provided by these languages are not in total adequate to describe highly dynamic processes on mobile distributed computing systems [18].

Closest to the required concepts as mentioned above is the meta-model language *XPDL* [10], which was developed as an abstract interchange format for different workflow engines. It provides a very general view on processes, is open for extensions and ready for all kind of automated and manual services. On the other hand, due to its high level of abstraction, it does not provide sufficient concepts to perform distributed process execution and handle errors as well as transactions.

In contrast, *BPEL4WS* [1] as a language for the orchestration of activities defined as web services, offers very specific and powerful elements to link tasks and to deal with unexpected circumstances as well. Processes defined with

BPEL4WS are ready to be executed but limit cooperations between business partners using the Web Service protocol stack. Furthermore, process descriptions tend to become rather complex due to possible combinations of sequential blocks with graph-structured elements in order to express parallel behaviour. Again, the definition language is developed for running on a central workflow engine and does not provide concepts for distributed process execution.

The *Web Service Choreography Interface* (WSCI) [2] is an add-on of WSDL and concentrates on the choreography of web services by describing a task from the individual perspective of its participating services. Therefore, the description itself is lean because each one is intended for only one single participant. The disadvantage of *WSCI*, however, is that all possible participants have to be determined in advance so the processes' information can be distributed and a fixed compatible interface can be implemented within the *WSDL description* of each participant. Also dynamic processes or ad-hoc workflows as well as often changing vicinities of mobile devices cannot be handled with *WSCI*.

A very lean description language is provided by JPDL [8], which is an integral part of the *Java Business Process Management (JBPM)*. *JPDL* supports manual tasks, but the description of automated function logic is matched to the Java programming language and the composition of web services is not provided at all. For error handling, *JPDL* also relies on the JAVA platform and, therefore, cannot be considered to be totally platform-independent.

EbBPSS is the *Business Process Specification Scheme* of the *EbXML* framework [11]. In particular, it is designed to describe business transactions and therefore it focuses on the aspect of binary collaboration between several companies. Although *EbBPSS* has the ability to describe quality and security issues as fixed requirements for the scheduled cooperation, it depends highly on the ebXML framework which is in itself too complex for most of today's mobile computing systems. Standing alone, it does not support the description of required control flow constructs, such as error handling mechanisms or the possibility to integrate users and different kind of services.

So, in summary, none of the considered approaches supports transfers of process descriptions and allows a completely distributed administration of mobile processes. Late binding of participants is often possible, but there are no adequate concepts to choose participants by their respective quality or by other non-functional criteria. In most cases, the description of activities and their dependencies within the process is very extensive or requires a lot of computing power to work on it. This, however, is not suitable for relatively weak mobile devices. Finally, concepts for handling faults are insufficient for the error-prone mobile computing systems and the handling of connection resets and security issues has not been considered at all since these process description languages have been developed basically for reliable central workflow engines.

3 A Mobile Process Integration Service

These deficiencies of already established approaches for describing mobile processes (cp. section 2) adequately motivate the development of an enhanced description language which fulfils all of the specified requirements. Accordingly, this section presents relevant features such an approach based on (*a*) a process description language for distributed processes and (*b*) a corresponding mobile process execution engine. But as such an engine cannot be realised without an underlying system infrastructure, subsection 3.1 first provides an outline of the middleware architecture as developed for that purpose in the DEMAC project.

3.1 A Middleware Architecture for Supporting Distributed and Mobile Processes

The decision to design a tailored system infrastructure for supporting a seamless integration of mobile processes into a mobile computing middleware evolved from an analysis of the processes' requirements and the respective features as offered by existing middleware approaches. Especially the close cooperation between the mobile processes and the context model to distribute and execute the processes lead to the need of a specifically adjusted model and service architecture.

The resulting system architecture is based on four basic service components (see figure 1) which are briefly described overview before section 3 introduces the integration of mobile processes in more detail.

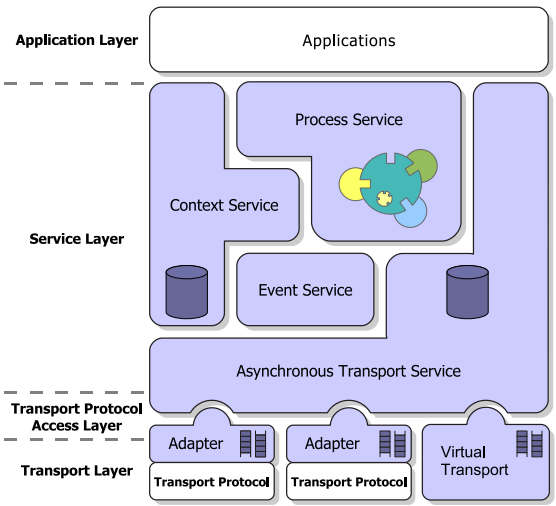

Figure 1. The DEMAC Abstract Architecture

The Communication Basis The *asynchronous transport service* and the *event service* form the communication platform of the architecture and provide communication with both push and pull semantics. This service abstracts from concrete transport protocols – like TCP/IP, Bluetooth or IrDA. To be independent from the underpinning protocols, the transport service uses its own addressing schema. These addresses are bound to a device and translated into concrete protocol specific addresses by the transport service. If the device is reachable by different protocols, non-functional aspects, like e.g. quality of service attributes, can be used to make an optimal choice.

The Context Service The *context service* collects and maintains all information about the context of the device. It acquires its knowledge either by events from the event service or by direct message exchange using the transport service. Towards the entities which use the service, it filters and partitions the information and provides only the amount of data they need. These are next to quality of service parameters also information about reachable devices and their services, location parameters and data about other users and their identity. To acquire the context information, a *federated approach* is chosen. Every device provides only local context information. To get the overall context, the information of the devices in the environment is merged. To find and resolve devices and services in the vicinity, the context service contains a *distributed registry* which uses peer-to-peer mechanisms to obtain its knowledge.

The Process Service The *process service* realises the integration of process management into the DEMAC architecture. It is comprised of two parts: The first one is a *definition language* in order to describe the mobile process as well as the users' and applications' non-functional demands (cp. section 3.2). Using this language, an application is able to define a sequence of activities, intermediary results which must be achieved, and constraints for the execution. The second part of the service is an *execution engine* for process definitions. This unit resolves and executes processes (cp. section 3.3). It can either invoke the activities locally or delegate the process to a remote process service. When delegating a process, the description and all necessary data is transferred to the remote unit by use of the transport service. Thereby the process service relies on the information provided by the context service to find a device providing the needed service and to enforce the non-functional demands and constraints. The execution engine's architecture provides the ability to extend a compact core by plugging in functional modules to adapt to the capabilities of the underlying device.

3.2 DEMAC Process Description Language

The *DEMAC Process Description Language*[1] (DPDL) is an XML-based description language to integrate distributed long-time processes into mobile computing

[1] http://vsis-www.informatik.uni-hamburg.de/projects/demac/dpdl1.0.xsd

systems. *DPDL* follows the meta-description language *XPDL* [10] and inherits the structure and those constructs of XPDL which turned out to be suitable for describing mobile processes.

The basic idea of *DPDL* is to allow a distributed handling of the process over heterogeneous systems. An entire process may be passed on to another device to continue work on the process's tasks. So devices which are not capable of executing a particular task of the process can mark its latest execution state and search for other devices able to carry on at the position established so far. So, by sharing the potential of several mobile devices, this approach increases the likelihood of successful process execution - even under the (generally unstable) conditionals which are typical for mobile devices and applications.

Meta-model and Structure As shown in figure 2, the basic container for the DPDL process description and all its data is a *Package*. A *Package* contains at least a single *WorkflowProcess*, which holds all tasks to be worked on (*Activities*) and the control flow as a fixed sequence to execute these tasks. *Activities* can be atomic or can be grouped to simple reusable blocks (*Activity Sets*), to a sequence of activities to be executed as a *Transaction* or to a set of repeatable actions within a *Loop*. Furthermore, an activity can represent an entire *Subprocess*.

To integrate non-functional criteria, the *Package* can also contain definitions of requirements for service qualities or for quality aspects of devices or networks. These requirements are modelled as *Strategies* and can be bound to activities or to the entire process.

To deal with likely occurrences of errors and connection resets DPDL introduces *Exception Handlers* and *Connection Reset Handlers*. These elements refer to another set of activities which should be executed in cases where the normal execution fails.

The introduction of *ActivityReferences* allows reusing the description of activities within the process, for example as a part of several error handling descriptions. *ActivityReferences* are linked by *Transitions* to describe the processes' control flow. *ActivityReferences* are unique within the process. They contain all information which is relevant for the execution of the activity in dependence of its position in the control flow, such as references to participants, error handling and non-functional criteria.

State Concept The state of each single activity within the process is modelled as a property of its respective unique *ActivityReference*, so the execution state of an activity is well-defined and the progress in processing the activities is visible for every participating device at any time during execution.

Figure 3 shows the potential lifecycle of an *ActivityReference*. An *ActivityReference* is *inactive* if preliminary activities are not executed or conditions for the execution of the referenced activity are not checked yet. In case one or more of these conditions can not be fulfilled, the *ActivityReference* is set to the error state *skipped*. If these conditions evaluate to true or there are no conditions defined, the *ActivityReference* is set to the state *ready*. It may happen that a

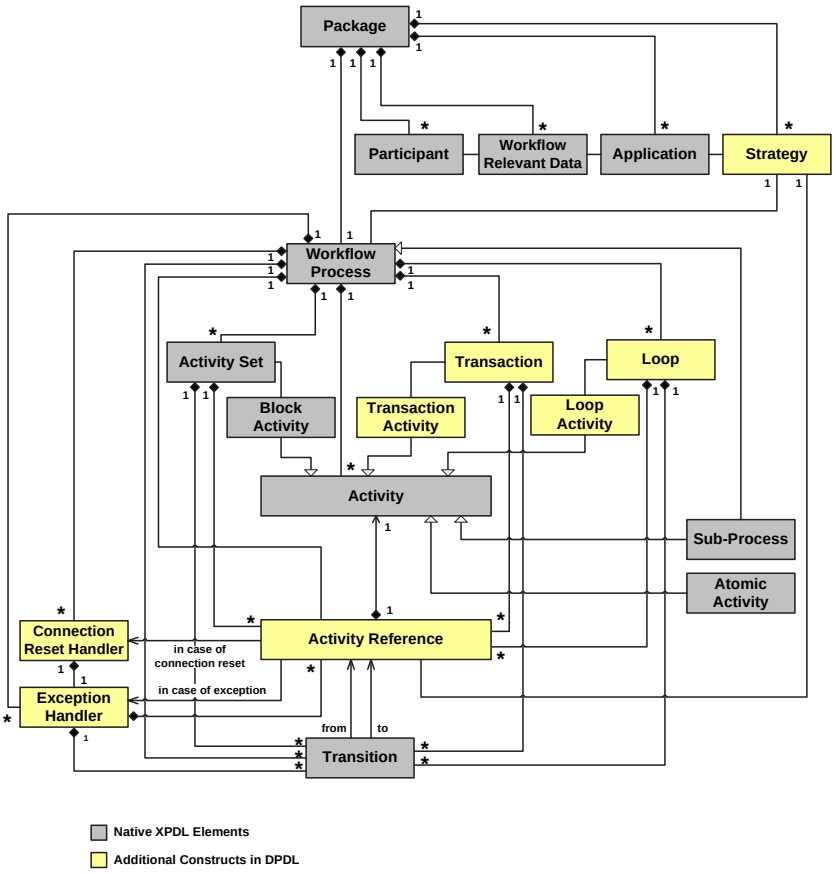

Figure 2. DPDL Meta-model

mobile device is capable of checking the conditions of an activity, but is not able to perform the execution itself. In this case, it will possibly take some time to transfer the process description to another device and it has to be checked close to the execution if the activity is still valid or if a defined expiration date is exceeded (error state *expired*). The states *skipped* and *expired* are also relevant for the appliance of a *Dead Path Elimination*. If all prerequisites are fulfilled and the actual execution starts, the *ActivityReference* is set to the state *executing*. The appearance of errors during the execution will result in a general error state *in error*. An activity is *executed* when its execution is successfully completed. It might now be set back to the *ready* state to be restarted later (for example if the activity is part of a loop) or it is set to the state *finished* which indicates the execution of the *ActivityReference* is terminated and finally closed.

Furthermore, a particular *ActivityReference* can be referenced as a start activity to mark the next task to be executed. This relieves other participating devices of dealing with tasks which have already been finished.

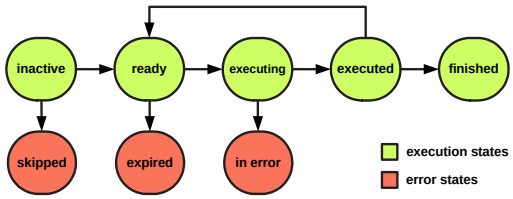

Figure 3. Possible States of Activities in DPDL

Description of Activities and External Data Transfer and execution of processes on mobile computing systems also require rather efficient use of the available amount of system memory. This means, one of the most important requirements of mobile processes is to make process descriptions as lean as possible. DPDL allows describing activities as a short but significant identifier and supports to store data external to the actual process. For example, huge documents may be kept completely out of the description until their processing time has arrived. This is particularly suitable if the data is needed only once or is used in very few activities within the process. On the other hand the provision of flexibility is essential in this case because the availability of devices and their connectivity may appear as a bottleneck to the dynamic integration of external features. So, it depends on the kind of application to decide whether or not obtaining data from a remote location.

Listing 1 shows the declaration of two variables by the use of the *DataField* construct and the definition of the corresponding data. While the content for the variable "PaintingName" can easily be hold within the process description for immediate access, the data item of the type "Image" is represented by an *ExternalReference* in order to save memory and network costs. Furthermore, the generic *Application* "Printer" is abstracted in the example listing by a *universal unique identifier (UUID)* which represents the category of adequate services to execute the respective activity, e.g. printing an image. The data involved in the task, in this case the painting's name and the image data itself, is finally called and mapped to the *Formal Parameters* of the generic *Application*.

Listing 1. Description of Data and Activities

```
<DataFields>
  <DataField Id="PaintingName">
    <DataType>
      <BasicType Type="String"/>
    </DataType>
    <InitialValue>Mona Lisa</InitialValue>
  </DataField>
  <DataField Id="NewPainting">
    <DataType>
      <DeclaredType Type="Image"/>
    </DataType>
    <ExternalReference Location="http://www.xyz.com/Very_Large_Image.bmp"/>
  </DataField>
</DataFields>

<Applications>
  <Application Id="Printer">
```

```
    <UUID>12345678901234567890123456789012</UUID>
    <FormalParameters>
      <FormalParameter Id="SomeName" Index="1" Mode="IN">
        <DataType>
          <BasicType Type="String"/>
        </DataType>
      </FormalParameter>
      <FormalParameter Id="SomePicture" Index="2" Mode="IN">
        <DataType>
          <DeclaredType Id="Image"/>
        </DataType>
      </FormalParameter>
    </FormalParameters>
  </Application>
</Applications>

...

<Activity Id="Print">
  <Implementation>
    <Tool ApplicationId="Printer">
      <ActualParameters>
        <ActualParameter>PaintingName</ActualParameter>
        <ActualParameter>NewPainting</ActualParameter>
      </ActualParameters>
    </Tool>
  </Implementation>
</Activity>
```

Users and Devices Mobile processes are highly related to tasks which require interaction with mobile participants such as users or devices or a combination of both. Therefore, special constructs are needed to describe which individuals are involved in which task and by what kind of communication channels these persons might be addressed or accessed. In *DPDL*, a participant is either totally specified or described in a generic way, e.g. by the declaration of a certain role. Descriptive properties of users (for example a digital identity) and devices (for example unique identifiers) can be combined to characterize a participant and help finding the required instance to execute the upcoming task (see listing 2).

Listing 2. Participants

```
<Participant Id="Smith" Name="John Smith">
    <Devices>
        <Device Id="111" Name="Personal Computer">
            <UUID>12345678901234567890123456789012</UUID>
        </Device>
        <Device Id="222" Name="Mobile Phone">
            <Devicetype Type="Cellphone"/>
        </Device>
    </Devices>
</Participant>
...
<ActivityRef Id="1" ActivityId="Activity1" ParticipantId="Smith" ... />
```

Handling Errors and Connection Resets Due to the high incidence of faults appearing in mobile computing systems, DPDL provides constructs to handle errors and unexpected connection resets. The description of *Exception Handlers* provides a definition of alternative control flow constructs to be executed when

an error occurs. In case of a connection reset, the communication may be either restarted, the service partner may be changed, or the activity may be skipped. The actual behaviour depends on the involved applications and the specific use-case and can also be modelled as a combination of activities (see listing 3).

Listing 3. Connection Reset Handler

```
<ConnectionResetHandler Id="1">
    <ExceptionId>someException</ExceptionId>
    <Retries>2</Retries>
    <NewSearch>true</NewSearch>
</ConnectionResetHandler>
...
<ActivityRef Id="1" ActivityId="Activity1" ConnectionResetHandlerId="1" ... />
```

Parallel Execution In case there is no relevant data dependency within the control flow, parallel paths of the process can be executed by different mobile computing systems. To share a process description, the responsible mobile device decides to execute an arbitrary parallel path and thereby sets its first *ActivityReference* to the state *executing*. While in this state, it produces a snapshot of the process description as a copy of its own process and forwards this copy to exactly one other device. Because the path chosen by the first device is already in the state *executing*, the second device can only select one of the remaining parallel paths.

In order to synchronize parallel paths, there has to be a defined meeting point, for example a stationary device. The participating devices can pass their copies of the process description to the given address. The service at the meeting point collects all incoming parallel paths belonging to the shared identifier and merges the copies to a single process description. If required, this one can be forwarded again to continue execution.

Modification of Activities In order to provide a maximum of flexibility, the description considers the possibility that activities may be modified throughout the execution of the process. For example, the single activity "Send a new text by e-mail" may be substituted by a more detailed *Activity Set* containing the two activities "Write text" and "Send e-mail". If no suitable service for executing the entire task can be found, other services may cooperate to compensate this lack of capability by executing intermediate steps. However, to control the amount of modification the initiator of the process can protect activities against unintentional changes by using suitable values for the Activity's *Editable* attribute. For example, the activity may be declared not editable at all, or the modifications might be further restricted by the definition of non-functional criteria, such that no semantically dependent activity can be substituted without compromising the overall correctness of the process.

The responsibility for exchanging or modifying activities resides with the context service which decides whether or not the upcoming task can be executed locally. The necessary knowledge about semantic equivalence of services and their

exchangeability or possible reconfiguration is kept by the distributed registry as part of the federated context services of all vicinal devices (cp. section 3.1).

Integration of Non-Functional Criteria To narrow the selection of potentially participating devices and services according to the user's interests and intentions, the process description may contain a set of non-functional criteria. The user who initiated a process can define a *Strategy* to assert a certain level of quality throughout the execution of the process. This way, *Strategies* help to ensure the user's goals as they were intended originally. Each *Strategy* contains a set of requirements which each hold a key-value-pair consisting of an identification argument and a target value. Listing 4 shows, exemplarily, how to define a limitation of the factor "cost" for the execution of a certain activity.

Listing 4. Description of non functional Criteria

```
<Strategy Id="123" Name="ActivityStrategy">
  <StrategyProperty Id="1" Name="Cost">
    <Requirements>
      <Requirement Name="MaxNetworkCost" Value="10"/>
      <Requirement Name="MaxServiceCost" Value="0"/>
    </Requirements>
  </StrategyProperty>
</Strategy>
...
<ActivityRef Id="1" ActivityId="TestActivity1" StrategyId="123"/>
```

Before executing an activity with specific requirements, the context service has to collect the relevant quality information, so the process service can ensure that only those services and devices are involved in the activity's execution which meets the specified requirements.

3.3 Mobile Process Execution

Depending on their intended purpose, mobile devices can have many different properties and a wide range of capabilities. To integrate most mobile devices and to benefit from the collaboration of heterogeneous systems, the mobile process execution engine must support different levels of performance.

Therefore, the execution engine is characterized by a modular design (cp. figure 4). A *Core Module* provides basic functionality such as receiving, storing, and forwarding process descriptions. It can be run independently on less powerful devices, like PDAs or cellphones, which do not provide enough memory or computing power to execute complex tasks but are useful to transport the process descriptions to other (different) environments. The core module also provides the interface for applications to initiate processes by passing the *DPDL* process description to the execution engine.

A more powerful *Base Module* is responsible for executing the described tasks of the process. It uses the core component to communicate with other devices and can be enhanced by further task-specific Extension Modules. *Extension Modules* are strongly dependent on the characteristics of the device, for example, an

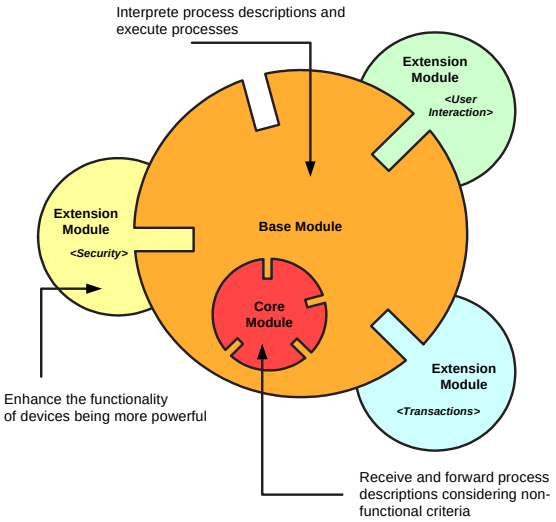

Figure 4. Modular Execution Engine for Mobile Processes

additional component supporting user interaction can only be realised if the respective device has a proper user interface.

The complete set of all installed components together with the DPDL description of mobile processes realises the *DEMAC process service*, which can have different combinations of execution modules, as shown in figure 4.

Finally, the mobile process execution engine cooperates closely with the *DEMAC context service* in order to get information about the device's vicinity, such as available services, environmental data or its own identity. If a new process description is received by the core module, the process data is made persistent and the process's *Strategies* are extracted from the *Package*. In case there is no base module attached or the proper component to execute the process locally is missing, the *context service* is requested to find a device suitable to the specified constraints to continue the execution. Otherwise, the execution engine within the responsible mobile device starts working on the process itself. It picks the upcoming *Start Activity*, examines it and requests the context service to find suitable services to process the task, depending on the defined *Participants*, *Strategies* and/or *Conditions* of this activity. If an adequate service for executing the upcoming activity cannot be found, the local execution engines marks the latest execution state, stops working on the process and again requests to find an alternative device to continue. This way, sharing the different properties and potentials of context aware mobile computing systems even complex and long-time processes can be executed in a step-by-step-manner.

4 Conclusion

This paper describes an approach to make mobile computing middleware platforms capable of supporting abstract descriptions as well as new execution models of *mobile distributed long-term business processes*. Due to (a) distributed and cooperative nature of such processes and (b) restrictions and specific characteristic of mobile computing environments, already existing description languages and execution models for centrally coordinated processes do not suffice. Therefore, an extended, technology independent *description language* is proposed and a corresponding *execution platform* and its realisation are described in this paper.

Thus, the paper presents the *DEMAC Process Description Language* which extends the XPDL meta-model by concepts for distributing and executing processes in mobile and frequently changing vicinities. It also describes the prototype realisation of an *execution engine* for such mobile processes. Thereby the paper argues that the presented modular design is able to support most of the heterogeneous capabilities of typical mobile devices.

As a prototypical implementation of the presented architecture has been realised already, future work includes implementation – on top of this platform – some of the project's use cases and sample scenarios. These include, e.g., a prototype of a *claim manager application* for an insurance company which creates customised mobile processes out of a template base and executes them using the DEMAC middleware. Furthermore, the overall performance of the system is continuously evaluated and improved. More fundamental questions arise in the fields of integrating privacy and security mechanisms as well as developing an adequate transaction concept for distributed and mobile processes.

References

1. Andrews, Tony and Curbera, Francisco and Dholakia, Hitesh and Goland, Yaron and Klein, Johannes and Leymann, Frank and Liu, Kevin and Roller, Dieter and Smith, Doug and Thatte, Satish and Trickovic, Ivana and Weerawarana, Sanjiva. Business Process Execution Language for Web Services Version 1.1. Specification, IBM, BEA Systems, Microsoft, SAP AG, Siebel Systems, 2003.
2. Arkin, Assaf and Askary, Sid and Fordin, Scott and Jekeli, Scott and Kawaguchi, Scott and Orchard, David and Pogliani, Stefano and Riemer, Karsten and Struble, Susan and Takacsi-Nagy, Pal and Trickovic, Ivana and Zimek, Sinisa. Web Service Choreography Interface (WSCI) 1.0. Specification NOTE-wsci-20020808, World Wide Web Consortium, 2002.
3. Braun, Peter and Rossak, Wilhelm. *Mobile Agents - Basic Concepts, Mobility Models, and the Tracy Toolkit*. Elsevier and Morgan Kaufmann and dpunkt.verlag, 2005.
4. Capra, Licia and Emmerich, Wolfgang and Mascolo, Cecilia. Middleware for Mobile Computing: Awareness vs. Transparency. In *In Proceedings of the 8th Workshop on Hot Topics in Operating Systems*, 2001. extended version.
5. Dey, Anind K. Understanding and Using Context. *Personal and Ubiquitous Computing Journal*, 5(1):4–7, 2001.

6. Dürr, Frank and Hönle, Nicola and Nicklas, Daniela and Becker, Christian and Rothermel, Kurt. Nexus–A Platform for Context-Aware Applications. In Roth, Jörg, editor, *1. Fachgespräch Ortsbezogene Anwendungen und Dienste der GI-Fachgruppe KuVS*, 2004.
7. Forman, Georg H. and Zahorjan, John. The Challenges of Mobile Computing. Technical Report TR-93-11-03, University of Woshington, 3 1994.
8. JBoss Company. JBoss jBPM 3.0 - Workflow and BPM made practical. Documentation, JBoss Company, 2005.
9. Leymann, Frank and Roller, Dieter. *Production Workflow - Concepts and Techniques*. PTR Prentice Hall, 2000.
10. Norin, Roberta and Marin, Mike. Workflow Process Definition Interface – XML Process Definition Language. Specification WFMC-TC-1025, Workflow Management Coalition, 2002.
11. Riemer, K. EbBPSS Business Process Specification Schema, Version 1.01. Specification, Oasis ebXML Business Process Project Team, 2001.
12. SAP AG. SAP Mobile Infrastructure: An Open Platform for Enterprise Mobility. Technical report, SAP AG, 2003.
13. Satyanarayanan, Mahadev. Fundamental Challenges in Mobile Computing. In *Proceedings of the Fifteenth ACM Symposium on Principles of Distributed Computing* , 1996.
14. Schilit, Bill N. and Adams, Norman and Want, Roy . Context-Aware Computing Applications. In *Proceedings of the* 1^{st} *International Workshop on Mobile Computing Systems and Applications*, pages 85–90, 1994.
15. Schmidt, Albrecht and Beigl, Michael and Gellersen, Hans-W. There is more to Context than Location. In *Proceedings of the International Workshop on Interactive Applications of Mobile Computing*, 1998.
16. Weiser, Mark. The Computer for the Twenty-First Century. *Scientific American*, 256(3):94–104, 1991.
17. Weiser, Mark. Ubiquitous Computing. *IEEE Computer Hot Topics*, 1993.
18. Zaplata, Sonja. Prozessintegration in Middleware für mobile Systeme. Master's thesis, University of Hamburg, 2005.