Mobile Processes: Enhancing Cooperation in Distributed Mobile Environments [*]

Christian P. Kunze, Sonja Zaplata, and Winfried Lamersdorf
Distributed Systems and Information Systems, Department of Informatics
University of Hamburg, Hamburg, Germany
Email: {kunze, zaplata, lamersdorf}@informatik.uni-hamburg.de

Abstract

Currently, context awareness is one of the main trends in distributed mobile computing environments. Against this background, the demand for more complex – and additionally long-term – mobile applications increases continuously. Nevertheless, most current available mobile applications – as well as their supporting middleware platforms – are still relatively monolithic and closed systems, concentrating on only short-term activities. As a consequence, most present appliances are still restricted to rather simple tasks and are therefore rather insufficient for more complex ones which consist of sequences of related long-term activities tied together in respective application-oriented processes. In order to overcome the resource and capability restrictions of mobile environments, such application processes may profit from, e.g., cooperation between devices in the mobile vicinity – a fact which is hardly supported by existing systems.

Therefore, this paper introduces a concept for integrating explicit support for mobile processes into mobile system infrastructures and for distributing their execution over different nodes in the network. Additionally, a corresponding middleware platform for context-aware and cooperative mobile applications is proposed. This framework has been designed and realized in the context of project DEMAC (Distributed Environment for Mobility-Aware Computing) which supports such migrating processes and helps to execute them under the restrictions imposed by realistic mobile applications. In particular, this paper proposes a corresponding process description language and an execution model for such mobile (business) processes.

1 Introduction

Compared to traditional distributed systems, mobile computing environments are characterized by additional requirements and limitations like, e.g., resource restrictions or increased variability in performance and reliability of wireless connections [19]. To overcome such constraints, mobile computing applications can take advantage of context information to break up their isolation by decreasing the distribution transparency and thus getting access to additional environmental data and services. Therefore, applications and middleware platforms evolved to become aware of their frequently changing vicinity and adapt and react to it accordingly and, thus, to benefit from their mobility.

Currently, so-called *context-aware* systems can be divided into four classes which describe how context information is used [4]: The first one is the class of *context-based selection* which contains applications that select their information or services based on context information, such as a tourist or museum guide in the *GUIDE* [7] respectively *Cyberguide* [1] projects. The applications of the second class – *context-based presentation* – adapt the way how they present their information according to contextual constraints. E.g. a mobile phone can switch from acoustic to optic notification of incoming calls during a meeting. The third class, which is called *context-based action*, includes applications that react to context changes (especially changes in the quality of used services) as they adapt the fulfillment of their own service accordingly. This is done e.g. in the *UbiQoS* project [5] where the frame rate of a video stream is adapted to the available bandwidth during its transfer. The fourth group of *context-based tagging or annotation* summarizes applications which provide the possibility to augment things or special context situations with additional information, such as in the *Stick-e Notes* project [16].

In summary, in such current systems 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. This means, contextual information about resources – like data, services or potential participants – is collected only once to create a tailored run of the mobile application. Because devices of the mobile vicinity do not cooperate in order to execute the application, available resources are determined and thus restricted by the capabilities of the executing device. Resources available by other devices remain inaccessible for the dynamic adjustment of the mobile application. As a consequence, also the complexity of applications and tasks

[*]This paper is based on "Mobile Process Description and Execution," by Christian P. Kunze, Sonja Zaplata, and Winfried Lamersdorf, which appeared in the Proceedings of the 6th IFIP WG 6.1 International Conference on Distributed Applications and Interoperable Systems (DAIS), Bologna, Italy, June 2006. © 2006 Springer.

is limited by the device‘s capabilities and its supported technologies. Hence, most existing middleware systems are rather application centric and, thus, offering assistance for basic but actually simple tasks. In order to come closer to the original vision of pervasive computing [22, 23], 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 utilize 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 a system platform that enables a new class of context-based applications which is called *context-based collaboration*. This is done by introducing the *Distributed Environment for Mobility-Aware Computing* (DEMAC) middleware framework – which realizes such an extension – with a special focus on a new description language and execution model for such *mobile (business) processes*.

The following subsections of the paper introduce the definition of mobile processes and provide a sample scenario to illustrate the concept of such 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. A more technical view on the prototypical implementation of the proposed middleware and on the evaluation environment is presented in section 4. The paper concludes with a summary and an outline of future work.

1.1 Integrating Processes into Mobile Computing Systems

The work presented here aims to extend the capabilities of mobile devices by means of 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 devices as process participants (cp. e.g. [18]). Accordingly, in our context, the term mobile process is defined and used as followed:

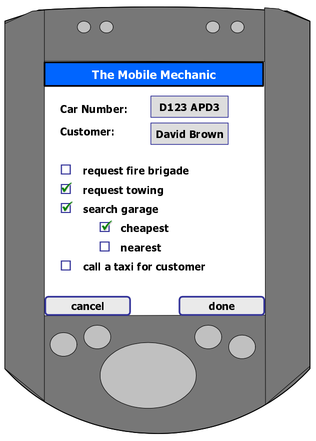
A mobile process is a sequence of (remote) services which may last over a longer period of time and span several devices during its execution. The results of the process are the effects the initiator expects from it.

In traditional mobile middleware, a process executes the application logic by explicitly assigning local or remote services to the process‘s activities and by invoking them directly. In addition 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.

Moreover, as some mobile process activities may last rather long (like hours, days, or weeks) the environmental conditions of the device can change dramatically between the executions of adjacent activities and even during the execution of single 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 its current 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 to execute the mobile process and then transfers the remaining process and its execution to one of them.

Considering the limited capabilities and resources of mobile devices, it is unlikely that one single device is able to execute a complex process entirely or that all required resources reside in its communication and capability range. Assuming that, e.g., a mobile device has to access a sequence of remote activities to obtain some data from them and that, in its direct vicinity, adequate services to perform these tasks exist, but that the mobile device cannot access them. The migration of the process provides the possibility that other devices may be capable to access the services and thus to fulfill the tasks. Furthermore, the migration opens up a new vicinity to search for other and maybe more suitable services.

As shown, 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 the likelihood of a mobile process to be executed successfully increases substantially.

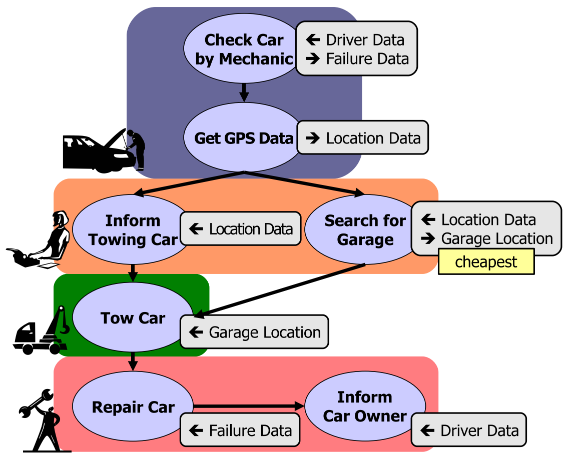

The Mobile Mechanic

Car Number: D123 APD3

Customer: David Brown

- ☐ request fire brigade
- ☑ request towing
- ☑ search garage
  - ☑ cheapest
  - ☐ nearest
- ☐ call a taxi for customer

(a) An Application to initiate Mobile Processes



(b) A Process to Recover a Broken-down Car

Figure 1: The Mechanic Assistant Application Example

1.2 Example: The Mechanic Assistant

To clarify the potential of mobile processes, a short example scenario is presented. In this scenario, a mechanic of an automobile club – who assists drivers of broken-down cars on the motorway – is supported by a pda-based service application (see figure 1(a)).

If the mechanic is not able to repair the car immediately he uses the mobile assistance application to initiate appropriate measures, like the towing of the car to a garage where the car will be repaired. Such required actions would cause a process of the following kind (see figure 1(b)):

> *After checking the car, the local position (e.g. data of the Global Positioning System) is determined to be able to notify a towing car and to search for an appropriate garage. To make an optimal decision the process is enriched with non-functional data, like the fact that the customer asked for the cheapest or nearest garage. The address of the selected garage is transmitted to the towing car which then carries the broken-down car to that garage. Finally, the mechanics of the garage repair the car and inform the owner that he can pick it up.*

In this example, the process is generated out of a template base by the application running on a PDA. During the execution of the mobile process, four roles are involved: the mechanic, the automobile club back office, the towing car, and the garage. These participants are resolved, in parts dynamically, into concrete services or servants at runtime. However, only by cooperation of these participants and the migration of the process among them, the expected results can be achieved.

Such a composed task, consisting of diverse – possibly distributed – subtasks, each of them depending on different environmental conditions and all of them forming a long-term process with manual and collaborative activities, is hardly supported by traditional middleware platforms so far.

1.3 Requirements for Descriptions of Mobile Processes

In order to describe processes in ways which allow 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* [14].

But in addition, they have also 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 run in order to save memory, CPU power, and energy resources. They also must 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 abstract notation of the desired services [10, 19].

Based on these requirements an analysis of related work and, in particular, existing process description languages are presented in section 2.

2 Related Work

This paper specifically concentrates on the description and execution of mobile processes. Therefore, an overview of related work in the area of mobile systems infrastructures and middleware platforms is given first – before existing approaches and techniques to express and execute long-term business processes are analyzed more extensively.

2.0.1 System Infrastructure

Since mobile process execution always relies on contextual information, the context modeling and context data acquisition are crucial for the respective concepts and system infrastructure to be developed. The abstract and generic definition of context and its data as used in the *Context Toolkit* [8] by Dey is mainly suited for the mostly a priori unknown demands of mobile processes. Whereas the understanding provided by Schilit [20] or Schmidt [21] 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 [9] to ensemble the context of an entity by federating local context clippings of entities within their particular vicinities 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* [6]. 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 about the way how this behavior is achieved. In addition, they do not have a *social behavior* 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 principle, similar requirements and, thus, solutions.

2.0.2 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 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 [25]. Table 1 summarizes the results of the following analysis in more detail and presents how the specified requirements are fulfilled by common process description languages, like *BPEL4WS*, *WSCI*, *JPDL*, *ebBPSS* and *XPDL*. Finally, *DPDL*, the proposed extended description language, is also validated against these requirements (cp. 3.2).

The *Business Process Execution Language for Web Services (BPEL4WS respectively WS-BPEL)* [2] is one of the most popular approaches to describe the orchestration of activities defined as Web Services. It offers very specific and powerful elements to describe activities, to link tasks and to deal with errors and transactions. Processes defined with BPEL4WS are ready to be executed but limit cooperations between business partners using the Web Service protocol stack and describing their services with the *Web Service Description Language (WSDL)*. To overcome these restrictions, various extensions have been discussed, as there are BPEL4PEOPLE [12] to cover human user interactions or BPEL-SPE to support the explicit definition of sub-processes [13]. Nevertheless, BPEL4WS process descriptions tend to become rather complex due to possible combinations of sequential blocks with graph-structured elements in order to express parallel behavior. Additionally, the definition language is developed for running on a central workflow engine and does not provide concepts for distributed process execution.

As well as BPEL4WS, the *Web Service Choreography Interface* (WSCI) [3] is an add-on of WSDL, but, in contrast to BPEL4WS, it concentrates on the choreography of web services by describing a task from the individual perspective of its participating services and thus in a distributed manner. 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 process‘s information can be distributed and a fixed compatible interface can be implemented within the WSDL description of each participant. Therefore, choreographies based on WSCI are hardly scalable, because the description of each existing partner has to be changed whenever a new participant is inserted. If, in addition, these partners are mobile and thus may become unavailable easily, dynamic processes or ad-hoc workflows as well as often changing vicinities of mobile devices cannot be handled with WSCI.

EbBPSS is the *Business Process Specification Scheme* of the *EbXML* framework [17]. 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 constructs, such as error handling mechanisms or the possibility to integrate users and different kind of services. Furthermore, the intricate extension of defined binary collaborations is inappropriate for dynamic mobile environments with more than two static participants.

A very lean description language is provided by JPDL [11], 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. For error handling, JPDL also relies on the Java platform and, therefore, cannot be considered to be totally platform-independent.

Closest to the required concepts as mentioned above is the *XML Process Definition Language (XPDL)* [15], which was developed as an abstract interchange format for different workflow engines. As a meta-model 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 pro-

Table 1: Analysis of Related Process Description Languages and DPDL

	BPEL4WS	WSCI	EbBPSS	JPDL	XPDL	DPDL
Service Composition	+	+	+	+	+	+
Late binding / service discovery at runtime	+	+	+	+	+	+
Support of migrating processes or distributed administration	-	+	-	-	-	+
Description of non-functional demands	-	+	+	-	-	+
Support of user interaction / manual tasks	-	-	-	+	+	+
Constructs to describe control flow	+	+	+	+	+	+
Constructs to describe data flow	+	+	+	+	+	+
Constructs to describe participants	+	+	+	+	+	+
Mechanisms for error handling	+	+	-	-	-	+
Scalability	+	-	-	+	+	+
Extensibility	+	+	+	+	+	+
Security support	-	-	+	-	-	○
Transaction support	+	+	-	-	-	+
Technology-independence	+	+	+	-	+	+
Audit Trail	-	-	+	-	+	+
Support of priority management	-	-	-	-	+	+
Economic consumption of memory	-	+	-	+/-	+/-	+
Economic consumption of computing power	-	+	-	+/-	+/-	+
Avoidance of communication overhead	+	+	+	+/-	+/-	+/-
Ability to handle connection reset	-	-	-	-	-	+

+ = supported - = not supported +/- = not applicable ○ = postponed

vide sufficiently powerful concepts to perform distributed process execution and handle errors as well as transactions. Since XPDL is released as version 2.0 [24], it is intended to be used as a file format for the graphical notation of BPMN. However, this enhancement does not affect the original purpose of XPDL and it does not introduce any beneficial concepts to support the given requirements of mobility either.

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

The deficiencies of established approaches for describing mobile processes (cp. section 2) motivate the development of an enhanced description language, called *DEMAC Process Description Language (DPDL)* which fulfills the specified requirements of the previous section (cp. table 1, last column). Accordingly, this section presents the most relevant features such an approach bases on: (*a*) the process description language for distributed processes and (*b*) a corresponding mobile process execution engine. But as such an engine cannot be realized 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 the given analysis of the process‘s 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 leads to the need of a specifically adjusted model and service architecture.

The resulting system architecture is based on four basic service components (see figure 2) which are briefly described before section 3.2 introduces the integration of mobile processes in more detail.

3.1.1 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.

3.1.2 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.

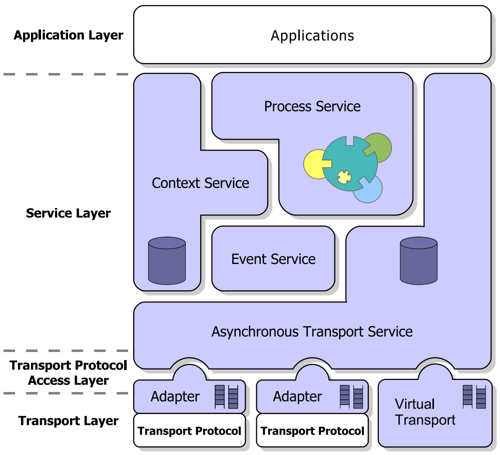


Figure 2: The DEMAC Abstract Architecture

3.1.3 The Process Service

The *process service* realizes 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 user‘s and application‘s 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, which resolves and executes mobile 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] is an XML-based description language to integrate distributed long-time processes into mobile computing systems. DPDL follows the meta-description language XPDL [15] 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 working 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.

3.2.1 Meta-model and Structure

As shown in figure 3, 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*.

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

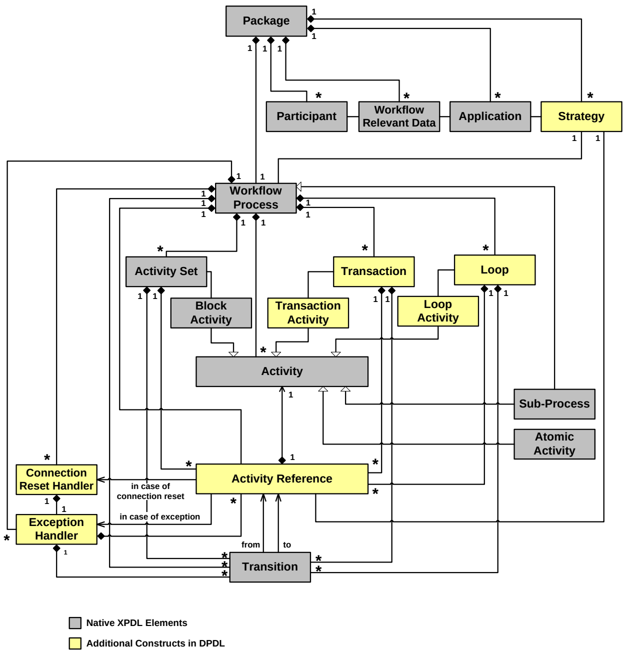


Figure 3: DPDL Meta-model

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 process‘s 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.

3.2.2 State Concept

In order to enable a distributed process to be executed whilst maintaining the autonomy of each single participant, three different cooperation scenarios can be distinguished: First, an activity can be executed *locally* by an application running on the same (mobile) device. If such local applications are unavailable, the device can search for adequate services on other devices in its vicinity to execute the task *remotely*. In case the direct vicinity does not provide the required service, the process description can be *transferred* to another remote device in order to enable the execution in a different vicinity.

The phase of migration can be considered the crucial point: All progress concerning the execution of the process has to be stopped and state information has to be documented in order to provide a consistent starting position to subsequently executing devices. In particular, the process cannot be transferred before all of the currently executed atomic entities are completed. This approach is advantageous because, as long as an activity can be executed locally, there is no need to search for another execution partner to accomplish this task. Furthermore, the process‘s consistency and integrity of its data must not be violated by splitting up atomic tasks.

Figure 4 shows the potential life cycle of a mobile process. On the basis of a common life cycle [14], it introduces the safe state *Option* in order to define a stable point to securely transfer a process during its execution. In detail, the *Option* state offers the possibility to either execute the process in case the upcoming activity can be executed locally, or to transfer the process description to another device.

At this central point the process life cycle embeds the potential life cycle of an *Activity* (c.p. figure 4). The process is regarded to be in the state *Running* if activities are being executed. The state of each single activity within the process is modeled 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.

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 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 next 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.

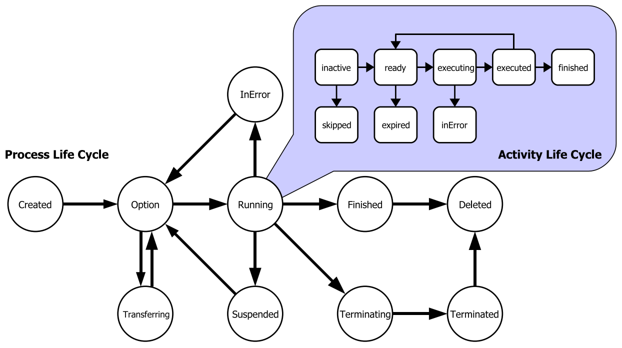


Figure 4: Process and Activity Life Cycles

3.2.3 Description of Activities and External Data

Transfer and execution of processes on mobile computing systems also require an 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 identifiers 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.

In the presented example (cp. section 1.2) the gathered GPS coordinates are important prerequisites for the execution of subsequent tasks. In addition, only a small amount of data has to be stored on (potentially mobile) devices. In this case, the content for the variable "Location Data" can easily be hold within the process description for immediate access. Transferring this data to a remote location would involve the risk of hazardous inaccessibility. In contrast, the provision of data resulting from the mechanics inspection might be very extensive, for example because it contains images or complex measured data. Furthermore, it is not necessarily needed to continue the process. Therefore, it would be adequate to store this data externally to the process description. In DPDL, the data item "Failure Data" can be represented by an external reference in order to save memory and network costs.

Considering the description of the task itself, the generic activity "Get GPS Data" is abstracted by a universal unique identifier (UUID) which represents the category of adequate services to execute the respective activity, in this case obtaining GPS coordinates of the current position. To resolve services into activities, an abstract service class is instantiated with an arbitrary service implementation. For instance, the customer's car navigation system might be able to offer a Web Service to provide GPS coordinates of its position. By ways of externalizing the semantic description for executing the task, the process description is relieved of complex details, e.g. the computation of GPS coordinates or satellite adjustment. If existing, the data involved in the task (service parameter) is finally called from the process description and is mapped to the formal parameters of the generic application. The output data of the resolved service, in this case the received content of the data item "location data" retrieved from the navigation system's Web Service is stored within the process.

3.2.4 Users and Devices

The integration of users and with them the introduction of human interactions is both necessary and beneficially. Mobile devices are primarily designed to accompany (mobile) users and therefore users should be addressable not only to initiate processes from mobile devices, but also to participate in the process manually and to solve unexpected errors by direct interaction.

Therefore, mobile processes are highly related to tasks which require interaction with mobile participants such as users or devices or a combination of both. 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 a unique identifier like the DEMAC transport address) can be combined to characterize a participant and help finding the required instance to execute the upcoming task.

In the example the automobile club's back office is a fixed participant to execute the activities "Inform Towing Car" and "Search for Garage". Therefore, the process description specifies how the back office can be addressed. In the scenario, this participant could be a stationary device which should be responsible for executing the mobile process. The device can be determined by an IP address or an arbitrary set of characteristics. Additionally or alternatively, a certain person in charge may be specified to manually supervise the process.

3.2.5 Handling Errors and Connection Resets

Due to the high risk of faults appearing in mobile computing systems, error handling mechanisms are essential for the execution of mobile processes. The main objective here is to provide a sufficient amount of information to be able to resolve most of the errors likely to occur and to recover from sudden connection resets and failures without depending on other resources. Therefore, the device currently responsible for the execution has to know unambiguously what to do in case the regular control flow fails. Even if the executing device itself encounters problems, it still must have the ability to recover its work from a save state or choose an alternative path to execute the process. Since alternative paths are totally dependent on the process‘s objectives, it is mandatory that the modeler of a process has the absolute flexibility to define what should happen in different erroneous situations.

DPDL supports the definition of user defined alternative paths and therefore provides special constructs to handle errors and unexpected situations. The description of *Exception Handlers* (cp. figure 3) 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 behavior depends on the involved applications and the specific use-case and can also be modeled as a combination of activities.

Just in case the responsible device collapses permanently, the execution will obviously fail. A possible solution to enforce fault tolerance in a rather pessimistic scenario is to integrate check points and to hold copies of the process description. For example, a parallel path can be executed on another device to control the work of its counterpart (see figure 5). If both paths cannot be joined before a specified deadline, the backup device will either carry on or initiate an error handling mechanism. Only if all of the possible recovery mechanisms fail, the user is involved in order to react to the acute problem.

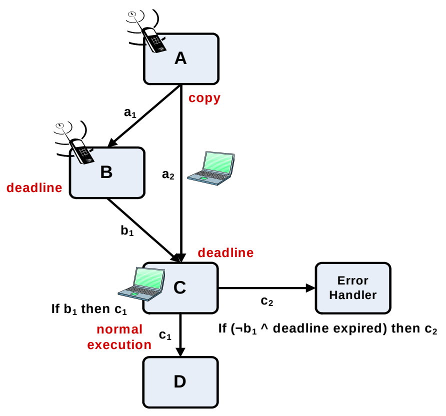


Figure 5: Error Handling: Guarded Execution

3.2.6 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. Using this strategy, there is always one device responsible for a specific path of the process description and it is therefore also responsible for error handling along this path.

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 a shared identifier and merges the copies to a single process description. If required, this one can be forwarded again to continue execution.

3.2.7 Integration of Non-Functional Criteria

To narrow the selection of potentially participating devices and services according to the user‘s interests, intentions, and demands the process description may contain a set of non-functional criteria. The user who initiated a process can define a *Strategy* (cp. figure 3) 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.

The scenario (cp. section 1.2) shows, exemplarily, how to define a limitation of the factor ”cost” for the execution of the activity ”Search for garage”. Assuming garages near to the current position determined by former activities offer electronic services to provide information about their details of repair, the parameter ”cost = cheapest” forms a strategy to select a service that fits the customer‘s interests best. From all available garages the one with the lowest costs will be picked accordingly.

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. Thus, services have to be comparable and have to offer information about their functional and non-functional properties in order to be considered for the selection procedure.

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 6). 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 cell phones, 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.

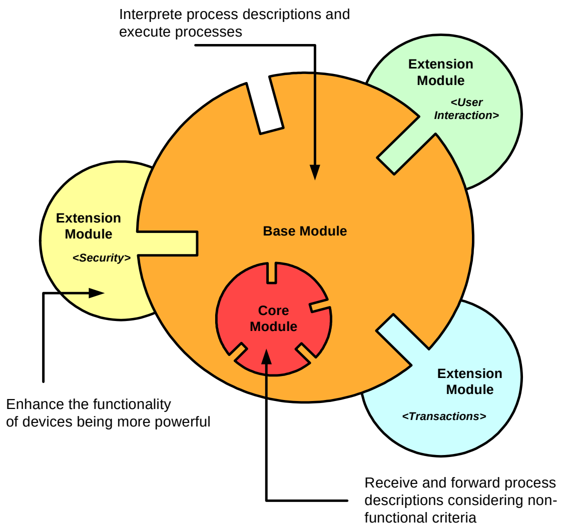

Interprete process descriptions and execute processes
Extension Module
<User Interaction>
Extension Module
<Security>
Base Module
Core Module
Extension Module
<Transactions>
Enhance the functionality of devices being more powerful
Receive and forward process descriptions considering non-functional criteria

Figure 6: Modular Execution Engine for Mobile Processes

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

The complete set of all installed components together with the DPDL description of mobile processes realizes the DEMAC process service which can have different combinations of execution modules (c.p.figure 6).

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 a 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 Prototype Implementation and Evaluation

The presented concept of a middleware for mobile processes (cp. section 3) has been prototypically realized for the system configuration shown in figure 7. The resulting system consists of several heterogeneous devices, operating systems and communication links which are: a Linux server, a desktop PC and a laptop with Windows XP, as well as an iPAQ Pocket PC with Windows Mobile 2003. The server and the desktop PC are connected through standard Ethernet and the Pocket PC communicates with the laptop using wireless LAN. These two separated networks are connected through a Bluetooth link between the laptop and the desktop PC.

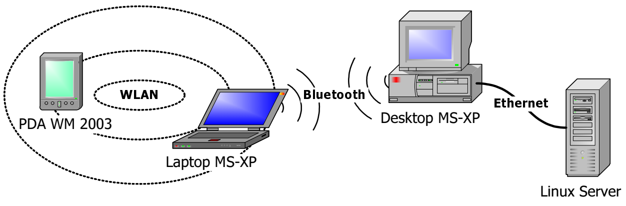


Figure 7: System Configuration of the DEMAC Prototype

For this system configuration, the DEMAC middleware has been realized twice, once for the *Java Platform, Micro Edition* with *CDC*[2] / *Personal Profile 1.0* and a second time for the *Java Platform, Standard Edition*. The communication over (wireless) LAN is realized with the integrated functionality of the Java environments and the Bluetooth link is realized with the Avetana Bluetooth Stack and respectively with the BlueCove Stack.

The transport system of the DEMAC platform – including transport and event service as well as the process service – has been realized with their main functionality. Thus the system is able to receive process descriptions from applications, to interpret processes, and to migrate them to other devices. Also the generic context model and the management system of the context service have been realized. Only the distributed registry and the generic service factory to create proxy objects for arbitrary services in the vicinity of the mobile device have not been finished so far (i.e. in fall 2006).

[2]Connected Device Configuration

Because of the stringent modular design, aiming at porting the middleware from Java ME to Java SE, only a reimplementation of the platform dependent components was necessary, such as the Bluetooth communication integration or XML-parser components. Using uniform adapter components to integrate different communication protocols, allowed for a dynamic and tailored configuration of the DEMAC platform for the different system components through easy to use property files.

In addition, the DEMAC middleware was also used and tested in academic education. By realizing context-enriched applications, such as, for example, a context-aware instant messenger, the suitability of the system for developing also more "standard" context-aware applications could be demonstrated.

5 Conclusion

In summary, the approach of mobile processes provides a basis to support the cooperation of mostly heterogeneous and a priori unknown devices. In this context, the paper described how to make mobile computing middleware platforms capable of supporting abstract descriptions as well as new execution models of *mobile distributed long-term business processes*. It was assumed that, due to (*a*) the distributed and cooperative nature of such processes and (*b*) the restrictions and specific characteristics of mobile computing environments, existing description languages and execution models for centrally coordinated processes do not suffice. Therefore, an extended, technology independent *description language* was proposed and a corresponding *execution platform* and its realization have been described in this paper.

The main contribution of this work is the definition of 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 realization 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. Overall, these concepts lead to a new class of context-aware applications which partly diffuse into the middleware and which are executed cooperatively. This new class is respectively called *context-aware cooperation*.

Finally, a number of additional issues remain to be addressed in order to evaluate and improve the overall performance of the system and to make the approach more practical. For instance, an adequate transaction concept for distributed and mobile processes has to be developed. At the moment it is being analyzed to which degree existent transaction models can be adapted to support the atomic execution of interrelated activities in highly unreliable systems. More fundamental questions arise in the fields of integrating privacy and security mechanisms into dynamic and ad hoc environments. The provision of trust is essential to achieve user acceptance, e.g. in order to allow private processes to be executed on foreign devices or to permit processes of others to be executed on their own devices. Therefore, concepts of encryption, authentication and privacy have to be surveyed and integrated as well.

As a prototypical implementation of the presented architecture has been realized already, a next step is to implement – on top of this platform – some of the project‘s use cases and sample scenarios to evaluate the architecture in more detail.

References

[1] G. D. Abowd, C. G. Atkeson, J. Hong, S. Long, R. Kooper, and M. Pinkerton. Cyberguide: A mobile context-aware tour guide. *Wirel. Netw.*, 3(5):421–433, 1997.

[2] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, and S. Weerawarana. Business process execution language for web services version 1.1. Specification, IBM, BEA Systems, Microsoft, SAP AG, Siebel Systems, 2003.

[3] A. Arkin, S. Askary, S. Fordin, S. Jekeli, S. Kawaguchi, D. Orchard, S. Pogliani, K. Riemer, S. Struble, P. Takacsi-Nagy, I. Trickovic, and S. Zimek. Web Service Choreography Interface (WSCI) 1.0. Specification NOTE-wsci-20020808, World Wide Web Consortium, 2002.

[4] C. Becker. *System Support for Context-Aware Computing*. Habilitation, University of Stuttgart, 2004.

[5] P. Bellavista, C. Stefanelli, and M. Tortonesi. The ubiQoS Middleware for Audio Streaming to Bluetooth Device. *mobiquitous*, 00:138–145, 2004.

[6] P. Braun and W. Rossak. *Mobile Agents - Basic Concepts, Mobility Models, and the Tracy Toolkit*. Elsevier and Morgan Kaufmann and dpunkt.verlag, 2005.

[7] K. Cheverst, N. Davies, K. Mitchell, and A. Friday. Experiences of developing and deploying a context-aware tourist guide: the GUIDE project. In *MobiCom '00: Proceedings of the 6th annual international conference on Mobile computing and networking*, pages 20–31, New York, NY, USA, 2000. ACM Press.

[8] A. K. Dey. Understanding and Using Context. *Personal and Ubiquitous Computing Journal*, 5(1):4–7, 2001.

[9] F. Dürr, N. Hönle, D. Nicklas, C. Becker, and K. Rothermel. Nexus–A Platform for Context-Aware Applications. In Roth, Jörg, editor, *1. Fachgespräch Ortsbezogene Anwendungen und Dienste der GI-Fachgruppe KuVS*, 2004.

[10] G. H. Forman and J. Zahorjan. The Challenges of Mobile Computing. Technical Report TR-93-11-03, University of Woshington, 3 1994.

[11] JBoss Company. JBoss jBPM 3.0 - Workflow and BPM made practical. Documentation, JBoss Company, 2005.

[12] M. Kloppmann, D. Koenig, F. Leymann, G. Pfau, A. Rickayzen, C. von Riegen, P. Schmidt, and I. Trickovi. WS-BPEL Extension for People - BPEL4People, 2005.

[13] M. Kloppmann, D. Koenig, F. Leymann, G. Pfau, A. Rickayzen, C. von Riegen, P. Schmidt, and I. Trickovi. WS-BPEL Extension for Sub-processes - BPEL-SPE, 2005.

[14] F. Leymann and D. Roller. *Production Workflow - Concepts and Techniques*. PTR Prentice Hall, 2000.

[15] R. Norin and M. Marin. Workflow Process Definition Interface – XML Process Definition Language. Specification WFMC-TC-1025, Workflow Management Coalition, 2002.

[16] J. Pascoe. The Stick-e Note Architecture: Extending the Interface Beyond the User. In *IUI '97: Proceedings of the 2nd international conference on intelligent user interfaces*, pages 261–264, New York, NY, USA, 1997. ACM Press.

[17] K. Riemer. EbBPSS Business Process Specification Schema, Version 1.01. Specification, Oasis ebXML Business Process Project Team, 2001.

[18] SAP AG. SAP Mobile Infrastructure: An Open Platform for Enterprise Mobility. Technical report, SAP AG, 2003.

[19] M. Satyanarayanan. Fundamental Challenges in Mobile Computing. In *Proceedings of the Fifteenth ACM Symposium on Principles of Distributed Computing* , 1996.

[20] B. N. Schilit, N. Adams, and R. Want. Context-Aware Computing Applications. In *Proceedings of the 1^{st} International Workshop on Mobile Computing Systems and Applications*, pages 85–90, 1994.

[21] A. Schmidt, M. Beigl, and H.-W. Gellersen. There is more to Context than Location. In *Proceedings of the International Workshop on Interactive Applications of Mobile Computing*, 1998.

[22] M. Weiser. The Computer for the Twenty-First Century. *Scientific American*, 256(3):94–104, 1991.

[23] M. Weiser. Ubiquitous Computing. *IEEE Computer Hot Topics*, 1993.

[24] WfMC. Process Definition Interface - XML Process Definition Language Version 2.00. Specification WFMC-TC-1025, Workflow Management Coalition, 2005.

[25] S. Zaplata. Prozessintegration in Middleware für mobile Systeme. Master's thesis, University of Hamburg, 2005.

Christian P. Kunze is currently a research assistant and Ph.D. candidate at the University of Hamburg, Germany. He received his diploma in informatics with a focus on distributed systems from the University of Hamburg in 2003.

He was software engineer at the *SHS Informationssysteme AG* before he became a research assistant. At the moment, he holds courses and advises projects in the field of mobile and pervasive computing, which is also the main focus of his research interests.

Sonja Zaplata works as a research assistant in the Department of Informatics, Research Group for Distributed Systems and Information Systems (VSIS), at the University of Hamburg, Germany. After finishing her training of Business Administration at the Wirtschaftsakademie Hamburg in 2000, she studied computer science and received her diploma in informatics from the University of Hamburg in 2006. Currently, she is a Ph.D. candidate and is interested in the research activities of distributed workflow management.

Winfried Lamersdorf is a full professor at the Department of Informatics at the University of Hamburg and head of the department as well as of the *Distributed Systems and Information Systems* unit there - with specific responsibilities in the area of distributed systems.

From 1974 to 1980 he studied computer science at the Technical University of Munich and the University of Hamburg. He also spent a year as a guest scientist at the University of Maryland, USA, in collaboration with the *National Institute of Standards and Technology* (NIST) in Washington/DC and received his doctorate for work in the field of database languages and semantic data models in 1985.

From 1983 to 1990 he was a staff-member in the *Distributed Applications* research group at the *IBM Scientific Centre* (WZH, 1983/84) and in the *IBM European Networking Centre* (ENC, 1984-90) in Heidelberg. There he started concentrating on Open Systems communication in general and communication support for database and distributed applications specifically.