Integrating Mobile Tasks with Business Processes: A Self-Healing Approach

Rüdiger Pryss, Steffen Musiol, Manfred Reichert
University of Ulm, Germany

ABSTRACT

Process management technology constitutes a fundamental component of any service-driven computing environment. Process management facilitates both the composition of services at design time and their orchestration at run time. In particular, when applying the service paradigm to enterprise integration management, high flexibility is required. In this context, atomic as well as composite services representing the business functions should be quickly adaptable to cope with dynamic business changes. Furthermore, they should enable mobile and quick access to enterprise information. The growing maturity of smart mobile devices has fostered their prevalence in knowledge-intensive areas in the enterprise as well. As a consequence, process management technology needs to be enhanced with mobile task support. However, tasks hitherto executed stationarily, cannot be simply transferred in order to run on smart mobile devices. Many research groups focus on the partitioning of processes and the distributed execution of the resulting fragments on smart mobile devices. Opposed to this fragmentation concept, this chapter proposes an approach to enable the robust and flexible execution of single process tasks on smart mobile devices by provisioning self-healing techniques to address the smooth integration of mobile tasks with business processes.

INTRODUCTION

Daily business routines increasingly demand a mobile and flexible access to information systems. However, the integration of smart mobile devices into existing IT infrastructures is laborious and error-prone. In particular, the IT infrastructure must cope with ad hoc events, various errors (e.g., connectivity problems), physical limitations of smart mobile devices (e.g., limited battery capacity), misbehavior of users (e.g., instant shutdowns), and environmental data collected by mobile sensors (Schobel et al., 2013). In general, proper exception handling constitutes a prerequisite for any mobile task support. In this context, adaptive and flexible process management technology offers promising perspectives based on a wide range of techniques (Reichert & Weber, 2012; Reichert & Weber, 2013; Kolb & Reichert, 2013; Lanz et al., 2013; Weber et al., 2008). In particular, it allows for the proper handling of run time exceptions. However, executing tasks on smart mobile devices in the same way as on stationary computers is not appropriate when taking these specific challenges of mobile environments into account as well.

Any service-oriented environment should allow for mobile task support during business process execution. This chapter presents an approach developed in the MARPLE (Managing Robust Mobile Processes in a Complex World) project. In particular, this approach enables the robust execution of single process tasks on smart mobile devices. Basically, it relies on two fundamental services, a backup and a delegation service. These services ensure that mobile tasks do not harm overall process execution in case task failures occur. Finally, a service-oriented architecture is presented that integrates the backup and delegation services with an existing process engine.

Note that implementing a process engine, which provides all functions for creating and executing mobile tasks, from scratch would constitute another option. However, if a process management system is already

in use in an enterprise, the introduction of another process engine might not be acceptable, due to the high efforts needed for transferring process models and their configurations to the new engine. Therefore, the presented approach provides an engine-independent interface for executing mobile tasks and services. For this purpose, the services are implemented in a layer between the process engine and the smart mobile devices. In particular, this service layer enables the instantiation, activation, and robust execution of mobile tasks (including error handling).
Furthermore, the *mobile task execution approach* presented in this chapter allows handling run time errors without need for any manual involvement of users. Generally, the provisioning of respective self-healing techniques is crucial for executing mobile tasks in the large scale as well as for achieving higher user acceptance for mobile business processes.
We first discuss fundamental issues relevant in the context of mobile environments. Their understanding is required for developing the delegation and backup services as well as for designing the overall system architecture. In particular, this architecture addresses the challenges (e.g., device failures) to be tackled in order to enable robust mobile task execution. Furthermore, these challenges must be considered in the context of business process execution; i.e., they must cover aspects beyond the characteristics of a mobile environment. In detail, the chapter addresses challenges posed by the mobile environment itself (e.g., a mobile device loses its connectivity), challenges related to process execution (e.g., missing data due to task failures), and challenges caused by the behavior of mobile users (e.g., instant shutdowns).

BACKGROUND

Many domains crave for the integration of smart mobile devices into business process execution (Pryss et al., 2012; Lenz & Reichert, 2007). Figure 1 shows a simplified healthcare example illustrating this. It depicts a ward round process for which mobile assistance is required (Pryss et al., 2012; Pryss et al., 2014). For instance, *Prepare Ward Round* constitutes a task whose mobile support would ease daily work of healthcare professionals.

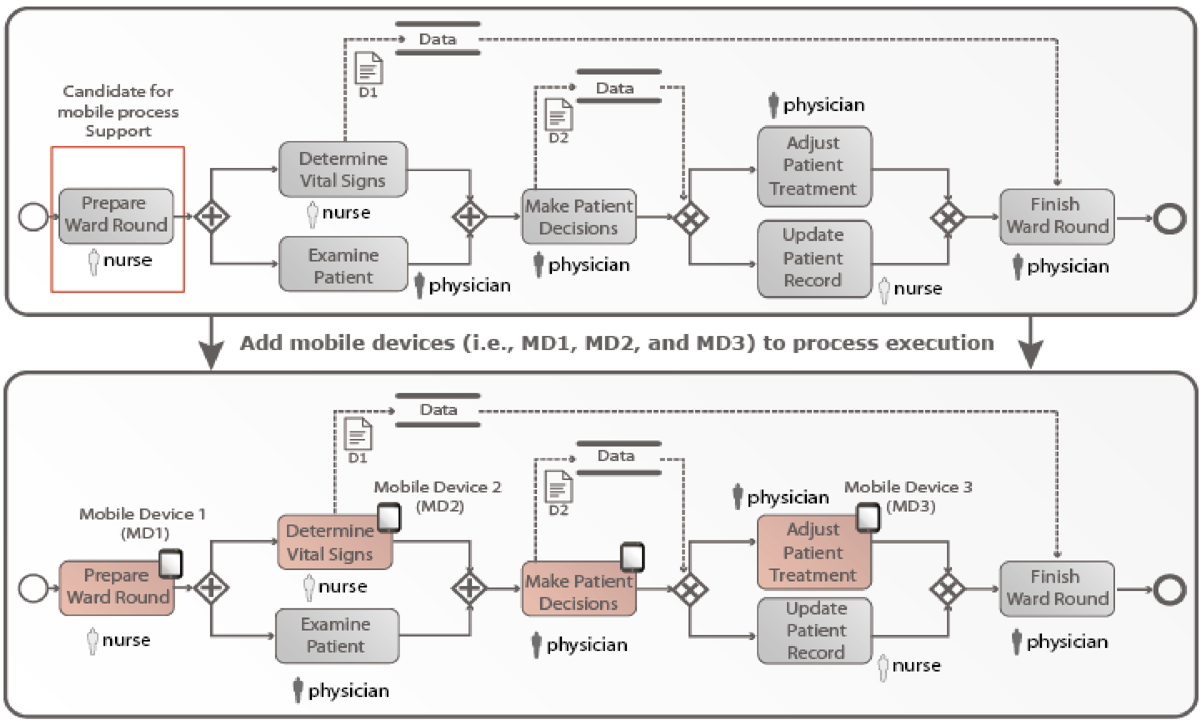


Figure 1. Adding smart mobile devices to process execution (mobile activities are flagged with an icon)

The use of smart mobile devices during process execution raises several challenges with respect to mobile task support. For example, if the smart mobile device running the task *Determine Vital Signs* (see Figure 1) encounters physical problems, overall process execution might be affected; or if tasks succeeding a mobile task in the flow of control require data usually provided by this mobile task, standard exception handling strategies (e.g., to skip the mobile task) are not appropriate if the mobile task fails. As shown in Figure 1, task *Finish Ward Round* is data-dependent on mobile task *Determine Vital Signs*. If task *Determine Vital Signs* fails at run time, the respective process instance cannot terminate properly since task *Finish Ward Round* cannot be properly executed due to the missing value of data element *D1*.

Challenges for Executing Processes with Mobile Tasks

To be able to run selected process tasks on smart mobile devices during process execution, the challenges imposed by mobile environments need to be properly addressed. These challenges concern the state of smart mobile devices as well as the behavior of mobile users. In addition, the specific challenges relating to process execution must be considered as well; e.g., dealing with missing process data due to run time failures of mobile tasks.
This section presents backgrounds on the elicitation of these challenges and categorizes them into process-, environment- and user-related ones.

Elicitating Challenges Relevant for Executing Processes with Mobile Tasks

In the following, three advanced mobile application scenarios will be discussed. The first scenario refers to the healthcare domain, the second one to the field of mobile augmented reality, and the last one to the psychological domain. The different scenarios focus on the challenges that emerge when executing mobile tasks. More precisely, only one scenario is directly related to business process execution, whereas the other two scenarios are not specific to business process execution; i.e., they refer to mobile application development in general and reveal fundamental challenges for mobile task execution. In particular, in all scenarios, two aspects are crucial: robustness and user acceptance.

Supporting Medical Ward Rounds with Mobile Task and Process Management

In hospitals, ward rounds are crucial for decision-making in the context of patient treatment processes. In the course of a ward round, new tasks emerge and need to be allocated to healthcare professionals. In current clinical practice, however, task management is not properly handled. Tasks emerging during a ward round are jotted down using pen & paper and their later processing is prone to errors. Furthermore, healthcare professionals must keep track of the processing status of their tasks and processes respectively (e.g., medical orders). To relieve healthcare professionals from such a manual task management, we developed the MEDo (MedicalDo) approach (Pryss et al., 2012; Pryss et al., 2014). It supports medical ward rounds by transforming the pen & paper worksheet to a smart mobile user interface on a tablet, integrating healthcare process support, mobile task management, and access to the electronic patient record. A number of user experiments has proven that MEDo puts task acquisition on a level comparable to that of pen & paper. Overall, physicians may create, monitor and share tasks based on the smart mobile and user-friendly platform provided by MEDo.
Prior to the development of MEDo, a case study was conducted in order to gain detailed insights into how ward rounds and task management look like in clinical practice. Based on the lessons learned, requirements for enabling a mobile task support in the context of medical ward rounds were derived. We further learned that current approaches, targeting at task and process support in hospitals, often neglect the way healthcare professionals organize their daily routine work. In addition, they do not provide the flexibility required by healthcare professionals to deal with exceptional situations (e.g., the health status of a patient changes from one moment to another) and they do also not allow for mobile task management.

In general, any mobile task assistance of physicians and nurses must cope with a number of challenging issues, e.g., frequent changes of a user's work location or connected network, mindless shutdowns of smart mobile devices, or non-observance of the battery charge level of a smart mobile device by its user. In turn, if a mobile task is interrupted and not properly resumed later, severe errors (e.g., omissive or missing data) might occur. Apart from this, tasks often have to be accomplished within a certain period of time, which is particularly challenging in a mobile environment. Table 1 summarizes the major challenges for executing processes with mobile tasks as identified in the context of the described mobile application scenario.

Location-based Mobile Augmented Reality Applications

Another challenging mobile application scenario is mobile augmented reality. In the AREA (Augmented Reality Engine Application) project (Geiger et al., 2013), we developed an advanced mobile application, which enables location-based mobile augmented reality on two different mobile operating systems (i.e., iOS and Android). In particular, this kind of mobile application is characterized by high resource demands since various sensors must be queried at run time and numerous virtual objects may have to be drawn in real-time on the screen of the smart mobile device. Therefore, as a particular challenge, AREA has focused on the efficient implementation of a robust mobile augmented reality engine, which provides location-based functionality. In turn, based on this engine sophisticated mobile business applications can be implemented. For example, one company uses AREA for its application *LiveGuide*[1]. A *LiveGuide* can be used to provide residents and tourists of a city with the opportunity to explore their surrounding by displaying points of interests stored for that city (e.g., public buildings, parks, places of events, or companies). In the latter context, we got detailed insights into the challenges related mobile application development in general (see Table 1).

Enabling Mobile Questionnaires and Data Collection in Psychological Studies

In psychology, many tasks and studies are performed with specifically tailored *paper & pencil* questionnaires. Usually, such a paper-based approach is accompanied by a massive workload when evaluating and analyzing the collected data afterwards, e.g., to transfer the data to electronic worksheets or statistics software. To relieve psychologists from such manual tasks and to improve the efficiency of data collection, we have developed smart mobile device applications in the QuestionSys[2] project for several psychological questionnaires (Ruf-Leuschner et al., 2013), (Isele et al., 2013), (Crombach et al., 2013). In this context, we were able to demonstrate the usefulness of smart mobile devices (e.g., smartphones or tablets) and mobile applications with respect to data collection in the psychological field. However, the smart mobile applications we developed have not been applicable to support psychological studies in the large scale yet. Moreover, during both their development and practical use in real-world psychological studies, we were able to identify numerous challenges. For example, certain psychological surveys might be performed in rural areas in developing countries (Crombach et al., 2013), which raises additional challenges regarding smart mobile applications and mobile (see Table 1).

Major Challenges for Executing Processes with Mobile Tasks

Regarding the execution of mobile tasks in the context of business processes, we consider the challenges discussed in the following. Thereby, we categorize them into challenges related to the environment, to the user behavior and process execution. Finally, Table 1 indicates in which of the presented mobile application scenarios these challenges could be identified.

[1] Further information to the project can be found at: http://www.liveguide.de or www.area-project.info
[2] Further information to the project can be found at: http://www.dbis.info/questionsys

Challenge 1: Connectivity (Environment related)
Connectivity refers to the availability of users as well as the smart mobile devices assigned to them. Non-availability might be caused by an undesired status of a device (e.g., broken device) or a particular personal status (e.g., user on vacation). In general, only if a smart mobile device is connected to a network, it should be considered as a target device for executing mobile tasks.

Challenge 2: Low Battery (Environment related)
A smart mobile device having a low battery status should not be considered as target platform for executing a mobile task until its battery is recharged, i.e., a low battery status indicates that the device (and its user) shall not be considered at the moment.

Challenge 3: Instant Shutdown (User behavior related)
In practice, users might instantly shut down their smart mobile device without reflecting on the consequences. Usually, this constitutes a short-term problem and the device will be restarted soon in most cases. However, if a user exhibits many instant shutdowns, this misbehavior should be taken into account. Note that instant shutdowns can be determined as follows: if a mobile device gets offline and then recovers, it is determined whether or not an instant shutdown has been the reason for this. For example, regarding Android devices, the shutdown action[3] has to be determined in this context.

Challenge 4: User Location (User behavior related)
At run time, a user might execute a mobile task at different locations. To improve mobile task execution, the location information is used for optimizing task assignment; i.e., assigning tasks, if possible, only to those mobile users, whose location match with the location defined for mobile tasks (see Challenge 6).

Challenge 5: Data Consistency (Process related)
Data dependencies between the different tasks of a process result from the order in which the respective process steps (i.e., activities) read and write process data elements. In particular, an activity might be data-dependent on another one that is realized as a mobile task.

Challenge 6: Location (Process related)
Data or physical objects needed to accomplish a mobile task might be only available at certain locations. If users are accomplishing their work, while continuously moving, it cannot be guaranteed that they will be at the right spot to gather the data needed.

Challenge 7: Urgency (Process related)
The urgency (i.e., priority) of mobile tasks needs to be considered as well. For example, if a lab test is required in the context of an emergency surgery, the urgency of the task performing this lab test will be high and the mobile task must be finished within a specified period of time after allocating the task to a mobile user**.**

Project	***C1***	***C2***	***C3***	***C4***	***C5***	***C6***	***C7***
MEDo	✓	✓	✓	✓	✓	✓	✓
AREA	✓	✓	✗	✓	✗	✓	✗
QuestionSys	✓	✓	✗	✗	✓	✓	✗
| (✓) : relevant | (✗) : not relevant |							

Table 1. Considered mobile application scenarios revealing challenges for executing processes with mobile tasks

[3] <action android:name="android.intent.action.ACTION_SHUTDOWN"/>

In the following, we present three alternatives for realizing processes with mobile tasks together with respective process models (see Figure 2):

Approach 1 *(Physical Process Fragmentation aka Process Partitioning):*
A process (i.e., process schema) is physically partitioned at design time. The resulting process fragments and their activities (i.e. tasks) are then assigned to different smart mobile devices before process instance creation time (and on an instance-per-instance basis, see Approach 1 in Figure 2). As a consequence, the execution of process fragments is spread over several smart mobile devices and hence must be synchronized during run time. In general, this raises specific challenges. For example, if the same data element is written by tasks from different process fragments, synchronization techniques are required to hand over data between the fragments and to ensure data consistency. In this context, a particular challenge emerges if a device encounters physical problems (e.g., a lost connection).

Approach 2 *(Logical Process Fragmentation aka Migrating Processes):*
A process schema is partitioned logically; i.e., it is split into logical process fragments that shall be executed on different smart mobile devices. As opposed to Approach 1, however, the original process schema is entirely known to each smart mobile device during run time (see Approach 2 in Figure 2). In the context of Approach 2, migration techniques are applied when completing the execution of a process fragment and handing over control to the next device (Zaplata et al., 2010(a); Zaplata et al., 2010(b)). In this context, the knowledge about the process schema is utilized to determine the next device and to transfer required data to it. Generally, it is determined at run time, which device shall execute which process fragments. In particular, this allows for dynamic exchanges of devices already assigned to a fragment. A specific challenge emerging in the context of Approach 2 concerns the synchronization of parallel process branches that may be concurrently executed on different smart mobile devices; i.e., a synchronization method is required to cope with data inconsistencies when joining the execution of the different branches.

Approach 3 *(Single Mobile Task Handling):*
Single process tasks are executed on smart mobile devices. For this purpose, a smart mobile device must cover a subset of the functionality of a stationary process client, e.g., a worklist component, which is continuously updated by the process engine (see Approach 3 in Figure 2).

Comparing the Approaches

Although all described approaches target at the support of mobile processes and tasks, respectively, they show subtle differences, which need to be carefully considered in the context of any implementation. In the following, we discuss and compare the three approaches. First, we discuss the differences between Approaches 1 and 2 and present work related to them. In particular, note that both approaches have been originally proposed without having a mobile application context in mind. Second, we discuss existing work on Approach 3. In the latter context, we do not relate Approach 3 to Approaches 1 and 2 due to their different goals.

Comparing Approaches 1 and 2: The basic goal of both approaches is to distribute the execution of a business process over several server machines in order to increase overall scalability of the process-aware information system. In literature, a multitude of techniques have been proposed in this context. Regarding Approach 1, for example, (Alonso & Schek, 1996) showed how distributed process execution may benefit from results known from distributed databases. In the sequel, different techniques for partitioning process schemas as well as for executing the resulting process fragments in a distributed way were proposed. For example, respective techniques were based on Petri nets (Aalst & Weske, 2001), statecharts (Wodtke & Weikum, 1997), or transactional dependencies (Alonso et al., 1996). All these approaches have in common that the overall process schema is partitioned into several process fragments, and the latter are then deployed to different execution devices (i.e., execution environments). As a particular challenge, a choreography protocol must be implemented to synchronize the execution of the different partitions (i.e., process fragments). In turn, to properly synchronize the partitions, the execution devices must be determined before process instance creation time. Furthermore, choreographies are complex to handle (Barros et al., 2005; Martin et al., 2008; Knuplesch et al., 2012; Knuplesch et al., 2013). Furthermore, the device executing a particular partition, only has knowledge about the schema of this specific partition. Hence, if changes of a partition become necessary, another protocol is needed to propagate these changes to other executing devices as well (at least if the change has effects on the synchronization of the distributed partitions).

In general, solutions referring to Approach 1 pose several drawbacks. To address the latter, techniques for migrating process instances between different sites and executing machines respectively (i.e., Approach 2) have been proposed (e.g., Cichocki & Rusinkiewicz, 1998). The respective sites, in turn, offer services that allow performing parts of the overall process model. These parts are often denoted as logical partitions. A process instance migrated between different sites carries all data produced by current or previous sites executing it (e.g., information about which site performed which tasks, about the data produced by the sites, and so forth). During process execution, the process instance is migrated from site to site. Thereby, each site executes parts of the process instance and gathers respective instance data. Since a site has always knowledge about the overall process schema and a process instance maintains all run time data related to it, coordination efforts in the context of process changes can be significantly reduced compared to Approach 1. As another benefit of Approach 2, the sites or execution devices can be determined and changed at run time. (Cichocki & Rusinkiewicz, 1998) discusses additional benefits of Approach 2; e.g., due to the rather small size of process schemas compared to the instance data determined at run time, migrating process instances between different sites causes less network traffic compared to Approach 1 (i.e., physically partitioned process schemas). As another benefit the process parts executed by the different sites (i.e., the granularity of distribution) may be determined during run time, i.e., after a site has finished its work, the subsequent site can dynamically determine what tasks it actually will execute. Note that in Approach 1 this has to be determined already at design time. Migrating a process instance between different sites also has some drawbacks. Since every site has full access to the entire instance data, privacy issues are more difficult to handle. Another shortcoming concerns tasks that have to be processed in parallel at different sites. In this context, a coordination protocol must be provided to properly synchronize the data concurrently produced at the different sites, i.e., for merging the data created concurrently at the different sites. Finally, ADEPT$_{\text{distribution}}$ (Bauer et al., 2003; Reichert et al., 2009) introduced advanced concepts for coping with the parallel execution of different process partitions at different sites, including the handling of dynamic process changes (Reichert & Bauer, 2007). Altogether, Approach 2 is promising in general and in the context of mobile process execution in particular.

Applying Approach 1 in a mobile context: There exist several proposals which apply Approach 1 in order to enable mobile process support. Many of these proposals use BPEL as underlying process modeling language; i.e., they allow partitioning BPEL process schemas and executing the resulting partitions (i.e., process fragments) on smart mobile devices (e.g., Baresi et al., 2004; Baresi & Guinea, 2007; Baresi et al., 2007; Schmidt & Hauck, 2007). In turn, (Philips et al., 2013) discussed the limitations existing in this context; e.g., the services called by the BPEL process must be known before run time, which is not appropriate for mobile environments. Recent research related to Approach 1 is presented in (Wakholi & Chen, 2012). Furthermore, there exist process-aware approaches relying on Web services and running on mobile devices (Battista et al., 2008; Hahn & Schweppe, 2009). In particular, they enable Web service flows on smart mobile devices based on a mobile process engine (Kunze, 2005; Schmidt et al., 2007; Stürmer et al., 2009). However, none of these approaches considers the characteristics of a mobile environment as presented above.

Applying Approach 2 in a mobile context: Approach 2 and its application to distributed mobile process execution have received growing attention recently. Several proposals exist that provide logical models for mobile processes (Hahn & Schweppe, 2009; Zaplata et al., 2009). Furthermore, advanced architectures and implementations of light-weight process engines were proposed (Hackmann et al., 2006). In previous work related to MARPLE (Pryss et al., 2010(a); Pryss et al., 2010(b)), for example, we showed how *Approach 2* can be realized. More precisely, we implemented a light-weight process engine that can be run on smart mobile devices and is able to interact with backend processes if required. In particular, the implemented mobile engine can execute an entire process schema or selected process fragments (i.e., logical partitions) autonomously on a smart mobile device. When developing this engine, basic concepts and design principles of the ADEPT process management technology were reused (Dadam et al., 2009; Reichert et al., 2009(b)). In particular, the ADEPT process meta model has been adopted as well as its fundamental correctness notions and verification procedures. Overall, the developed mobile engine is able to execute process fragments on smart mobile devices. Furthermore, the engine is able to deal with dynamic structural adaptations of process fragment instances (logical partitions) running on the smart mobile device; e.g., to cope with contextual changes in the environment or to handle exceptional situations (Dadam & Reichert, 2009; Reichert & Weber, 2012). Despite these commonalities with ADEPT, it is noteworthy that a complete new implementation of the kernel of the mobile process engine was required in order to meet the performance requirements of mobile application scenarios and to cope with the issues specific to mobile processes (e.g., small screen size). In particular, the implementation framework on which the engine is based differs from the one used in the context of the ADEPT project - ADEPT relies on JAVA, while the described mobile engine is based on the Microsoft .NET Compact Framework. Apart from the ability to run (and dynamically change) the schema of a process or process fragment autonomously on the smart mobile device, a particular need we identified was to properly address the characteristics of mobile environments. For example, scenarios like shutting down a device from one moment to the other must be considered. The same applies to scenarios in which mobile tasks can be executed more properly if location information is additionally used (e.g., a particular task may have to be performed in the laboratory). The work presented in this chapter constitutes the first important step towards a more proper integration of single mobile tasks with business processes. In future work, the results described in this chapter will be integrated with the work on the mobile process engine. Altogether, the presented mobile engine, we developed in the context of the MARPLE project, as well as the other proposals related to Approach 2, often neglect the characteristics of smart mobile devices on one hand and user acceptance on the other.

Approach 3: Note that Approach 3 focuses on the mobile enactment of single tasks of a business process. Therefore, we only discuss proposals that focus on proper execution of single tasks on smart mobile devices properly. For example, (Alonso et al., 1995) discussed the challenges of disconnected clients in the context of business process execution. However, no mobile context was particularly considered. Since disconnections of a (smart mobile) device on which a particular task shall be performed constitute a fundamental aspect of any mobile task execution, this work can be regarded as first proposal dealing with particular challenges of mobile task execution in the context of a business process. However, only little work, for example, (Tuysuz et al., 2013), exists which considers the proper execution of mobile tasks on smart mobile devices. For example, the problem of missing or inconsistent data is a predominant subject discussed in this context. (Hahn & Schweppe, 2009) proposed to apply transaction techniques to mobile processes. In order to handle failures of smart mobile devices (e.g., a disconnected device), the execution of mobile tasks is embedded in transactions. Furthermore, transactional properties are defined in order to determine in what cases a transaction must be aborted. However, the techniques provided by this and similar proposals do not deal with more complex failures as described in this chapter. In particular, they do not allow compensating failures, while ensuring the same execution semantics as originally intended. Another proposal relevant in the given context was made by (Pryss et al., 2012; Pryss et al., 2014), which describes the already presented MEDo approach. The latter deals with mobile task handling in the context of medical ward rounds, but does not consider the user-, process-, and environment challenges presented before.

There exist other proposals, which are not related to Approaches 1-3, but that focus on specific challenges of mobile environments like broken connections or lack of communication facilities. Combining real-time data and domain knowledge raises additional challenges for mobile process support and has been an intensive subject of research as well. For example, MobiHealth (Jones et al., 2006) supports context-aware services, which allow integrating sensor data. However, failure management of context-aware mobile processes is restricted in the sense that the execution of an instance may only be switched between pre-configured process variants in order to deal with contextual changes. Generally, approaches that allow switching between pre-configured process variants have been proposed by other authors as well (Hallerbach et al., 2010; Zaplata et al., 2009; Ayora et al., 2013). In particular, the handling of mobile task failures has not been considered by these proposals.

Altogether, Approaches 1 and 2 are predominant regarding research on distributed processes in general and distributed mobile processes in particular. However, the robust execution of mobile tasks (i.e., Approach 3) has not been sufficiently addressed so far. Therefore, this chapter focuses on *Approach 3* and, in particular, on the robust execution of mobile tasks, while at the same time not burdening mobile users at the occurrence of any errors. Note that *Approach 1* is not considered in the context of MARPLE due to its drawbacks (see above). In turn, Approach 2 is considered by MARPLE (see above), but will not be subject of this chapter. Figure 2 summarizes the presented approaches.

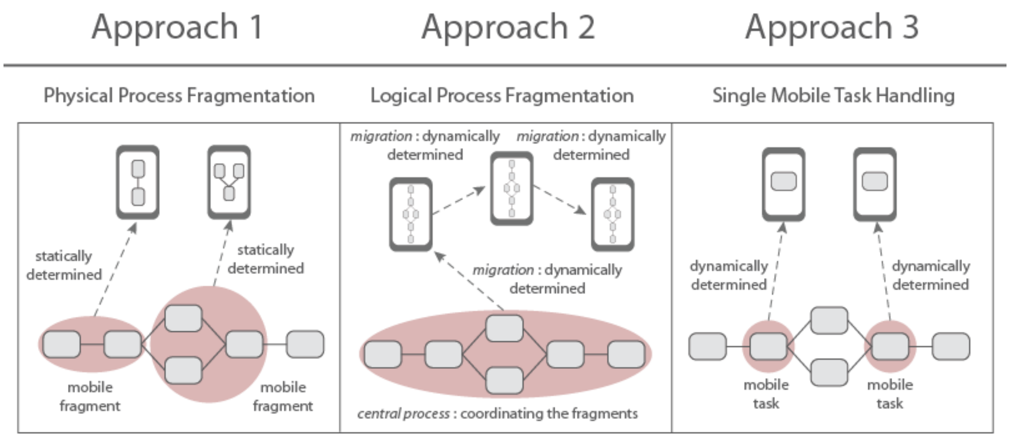

Approach 1
Approach 2
Approach 3
Physical Process Fragmentation
Logical Process Fragmentation
Single Mobile Task Handling
statically determined
statically determined
mobile fragment
mobile fragment
migration : dynamically determined
migration : dynamically determined
migration : dynamically determined
central process : coordinating the fragments
dynamically determined
dynamically determined
mobile task
mobile task

Figure 2. Approaches for realizing mobile processes

The next section presents basic concepts and services of the proposed MARPLE architecture enabling mobile task support.

MOBILE PROCESS TASK INTEGRATION ARCHITECTURE

This section presents an architecture for integrating mobile tasks with business process execution. Particular focus is on the realized delegation and backup concepts.

A Run Time Architecture for Mobile Tasks

Recall that the two fundamental concepts to deal with the above presented challenges are the backup service and a service for mobile delegation. During business process execution, the mobile delegation service ensures that already assigned mobile tasks are automatically re-delegated to another authorized

mobile user in case of errors. Finally, if no suitable mobile user can be determined, the backup service ensures that overall process execution will not be harmed.
Note that the delegation and backup services allow for the robust execution of mobile tasks. In particular, the design of these services allows for their integration with existing process engines.
Therefore, implementing a dedicated process engine that provides specific functions for creating and executing mobile tasks, constitutes one possible approach. However, if a process management system is already in use, usually, the introduction of another process engine will not be accepted (e.g., due to the high efforts required for transferring process models to the new engine). For this reason, the architecture presented in the following provides an engine-independent interface for executing mobile tasks. Since communication with Web services constitutes a core feature of any modern process engine, a service-driven approach has been realized (see Figure 4).

The core of the run time architecture for executing mobile tasks is denoted as *mobile execution environment* (see Figure 4). The mobile execution environment (MEE) extends existing business process environments that do not provide mobile task support. Thereby, it has components to manage mobile users (list management), to provide the delegation service as well as a component to provide the backup service. Furthermore, the MEE provides two important interfaces. First, an interface to integrate existing process engines with the MEE (see Figure 4, *pm interface*). Second, an interface to integrate mobile devices with the MEE (see Figure 4, *md interface*). Note that the *pm interface* is designed to integrate a wide range of existing process environments, e.g., Intalio (Bhandari et al., 2011), Activiti (Meister, 2011), or AristaFlow (Dadam et al., 2009).

In the following, base functions of the *pm interface* and the *md interface* (including the *mobile service client*, see Figure 4) will be shortly introduced. Their understanding is required for the following sections.

Process**M**anagement **Interface:**
The *pm interface* is designed to integrate the MEE with an existing business process engine. The main purpose of this interface is to exchange information with the process engine in order to execute mobile tasks. The MEE uses the interface to get a list of mobile tasks which shall be executed. These mobile tasks will be claimed by the MEE, which then takes the responsibility for them. Note that all actions (include failure handling; i.e., the delegation and backup services) to perform mobile tasks are handled by the MEE. Finally, after a mobile task has been finished, MEE uses the *pm interface* to return the computed data of the mobile task to the process engine.

Mobile**D**evice **Interface:**
The *md interface* is designed to integrate smart mobile devices with the MEE. These devices will be used to execute mobile tasks the MEE is responsible for. Note that the *md interface* has a component called *mobile service client*. The client is software which is deployed to smart mobile devices and provides functions to perform mobile tasks on one hand and to read smart mobile device properties (e.g., battery status) on the other. At present, the *md interface* provides the mobile service client for Android smart mobile devices. In future work, the client will be provided on Apple iOS and Microsoft Windows Mobile OS.

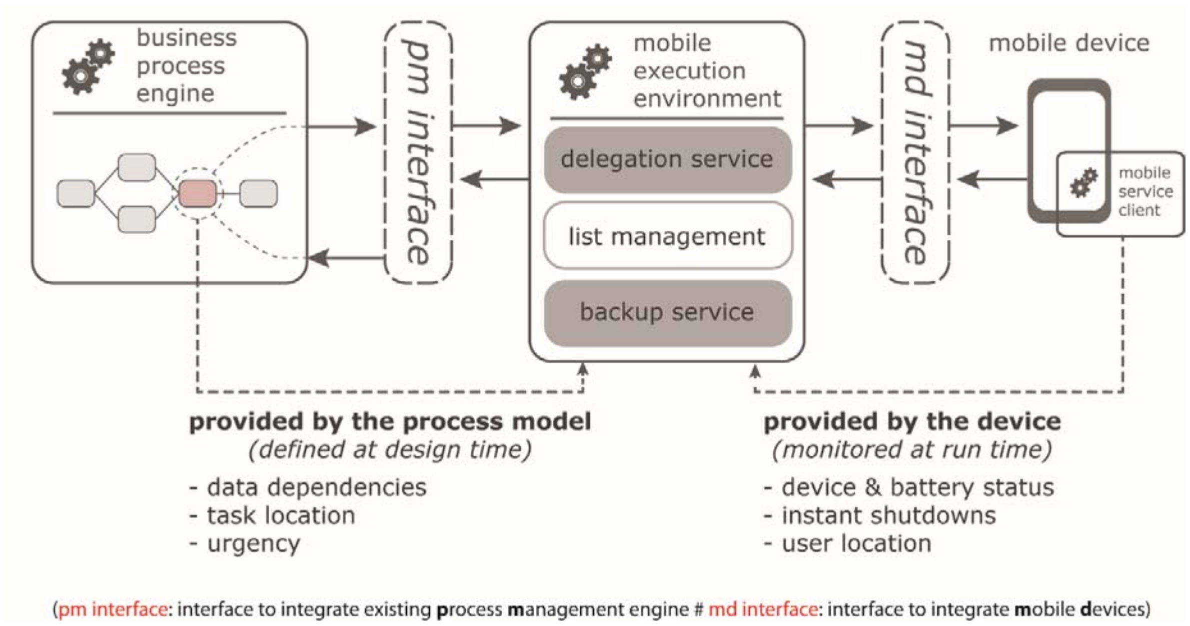
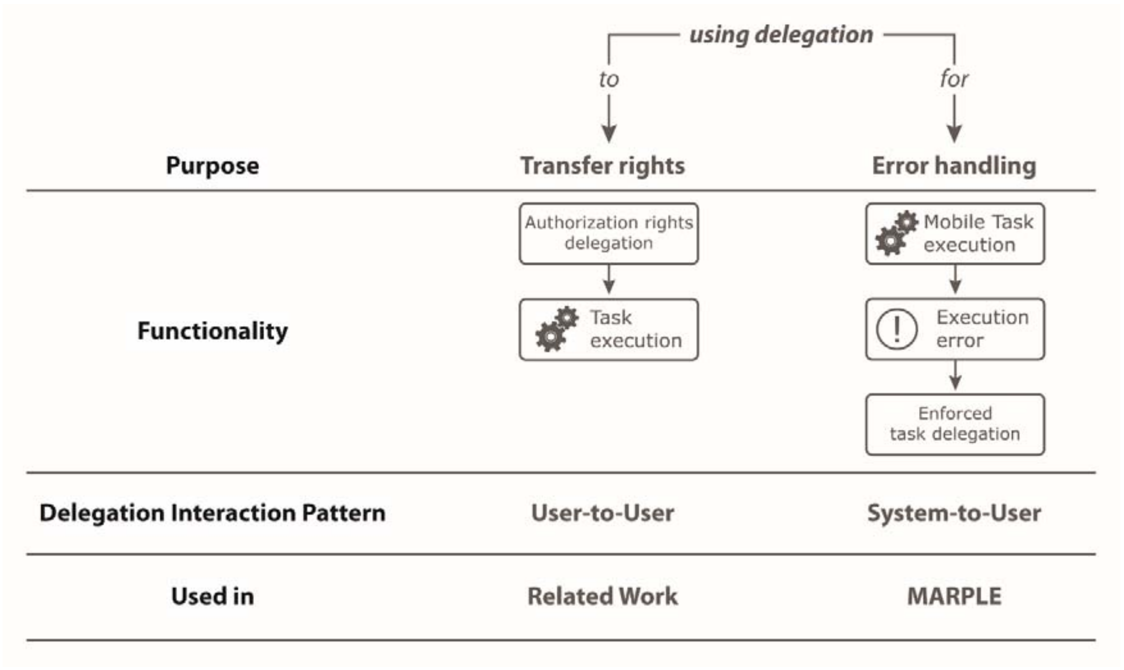


Figure 4. Mobile task run time architecture (simplified)

Task Delegation

Recall that during business process execution, the task delegation service ensures that already assigned mobile tasks are automatically re-delegated to another authorized mobile user in case of errors.
In the following, we introduce how task *delegation* is used during process execution in general and for our delegation service in particular (see Figure 5).


using delegation
to
for

Purpose	Transfer rights	Error handling
Functionality	Authorization rights delegation → Task execution	Mobile Task execution → Execution error → Enforced task delegation
Delegation Interaction Pattern	User-to-User	System-to-User
Used in	Related Work	MARPLE

Figure 5. Delegation concepts

Using Delegation to Transfer Rights:
Delegation permits a user to assign all or a subset of their authorization rights to other users not possessing these rights at the present moment (see the *functionality of Transfer rights* in Figure 5). Thereby, a user may assign these rights either in the context of a particular process instance or for all process instances. (Schaad, 2003; Crampton & Khambhammettu, 2008; Gaaloul & Charoy, 2009) provide various reasons for delegating respective rights; e.g., a user may not possess required documents, or an entailment constraint like *separation of duties* has to be enforced (Pryss et al., 2013).

Authorization rights for a particular task may be delegated either when a user is processing this task or before the task is assigned to any user. Usually, the first alternative is applied if delegation shall increase flexibility (e.g., another user has the required resources, but misses respective rights) or efficiency (e.g., another user is more efficient in performing the task) with respect to process execution. Applying the second alternative (i.e., delegating the task before assigning it to any user) will be useful if the authorized user who is usually performing the task is not available. Then, a delegation may have to be applied in order to determine other potential users processing the task.

In the given context, delegation is accomplished based on two techniques: First, delegation may be applied based on user-defined rules stored in a repository. Second, users may delegate their rights dynamically during run time. The two techniques can be summarized as *user-to-user* interaction pattern: A user determines the context in which a delegation may be applied. Finally, delegations only take place when a task is in a desired state. Note that delegation is usually used to transfer rights.

Using Delegation for Error Handling:
Delegation might also become necessary to handle errors; e.g., when mobile users or their devices encounter problems. This pattern is supported in the proposed architecture as well. Note that the delegation of a task may only be allowed if the target mobile user possesses the same rights as the mobile user who has encountered the problem. The delegation will then be performed based on a *system-to-user* interaction pattern. The mobile user being the target of the delegation is not allowed to decline the delegation request. In future work, we will optimize this pattern in order to also allow mobile users to decline delegation requests.

User List Management for Delegation and Backup

To foster robust execution of mobile tasks, three different user lists are maintained: an initial user list ul_{init}, a user list ul_{mob} comprising mobile users that match the criteria shown in Table 2 best for executing a mobile t_{mob}, and a delegation list dl_{mob} (see Figure 6). Note that these lists except ul_{init} are maintained by the MEE. ul_{init} is provided by a process engine and contains all mobile users u_{mob} authorized to perform a mobile task t_{mob}. Furthermore, ul_{init} constitutes the basis for creating the two other lists, which are determined based on an analysis of ul_{init}. Thereby, ul_{init} is created by considering the following properties:

Property	***Description***
Connectivity	Indicates whether a user u_{mob} is online or offline.
Low Battery Status	Indicates whether the user's device has a low battery status.
User Location (u_{mob}.location) vs. Task Location (t_{mob}.location)	The user's current location will be compared to the task location. For example, if the *location* attribute of a mobile task has value *emergency unit* and a mobile user is currently staying at another ward, he will not be considered for ul_{init} .
Pre-filter	Indicates whether u_{mob} has been excluded from ul_{mob} by a *pre-filter*.

Table 2. Properties for determining ul_{init}

As soon as mobile task t_{mob} becomes activated, ul_{mob} will be calculated by the *delegation service* as follows:

```
$ul_{mob} \leftarrow \{\}$
FOREACH $u_{mob}$ IN $ul_{init}$
    IF ($u_{mob}$.connectivity) AND ($\neg$ $u_{mob}$.low.battery.status) AND ($\neg$ $u_{mob}$.pre-filter)
        IF ($u_{mob}$.location = $t_{mob}$.location) or ($u_{mob}$.location = {} and $t_{mob}$.location=0)
        THEN
            $ul_{mob}$.append($u_{mob}$)
```

According to this procedure, all mobile users from ul_{init} being online, having a smart mobile device with sufficient battery status, not being pre-filtered (i.e., not being manually excluded for the process or an instance of the process), and located close to the required location of the task, will be appended to ul_{mob}. Finally, users contained in ul_{mob} will be notified about the mobile task that is ready for execution. As a result, this task will be added to their work list on the smart mobile device. After one mobile user has claimed the task, all other mobile users will be informed and, finally, the task be assigned. For this purpose, the work lists of all other users will be refreshed and the task be removed from these work lists.

Algorithm ul_{mob}: optional attributes u_{mob}.location and t_{mob}.location

Note that attributes u_{mob}.*location* and t_{mob}.*location* are optional. If the mobile task t_{mob} has no location (i.e. u_{mob}.*location* = ϕ) and the mobile user u_{mob} is not assigned to a location (i.e., t_{mob}.*location=0), the* algorithm to determine ul_{mob} does not add this mobile user to ul_{mob}. Note that the purpose of ul_{mob} is to assign tasks only to those mobile users being at the same location as defined for the respective mobile task. The remaining mobile users will added to the fallback delegation list dl_{fb}. The purpose of this list is explained in the following.

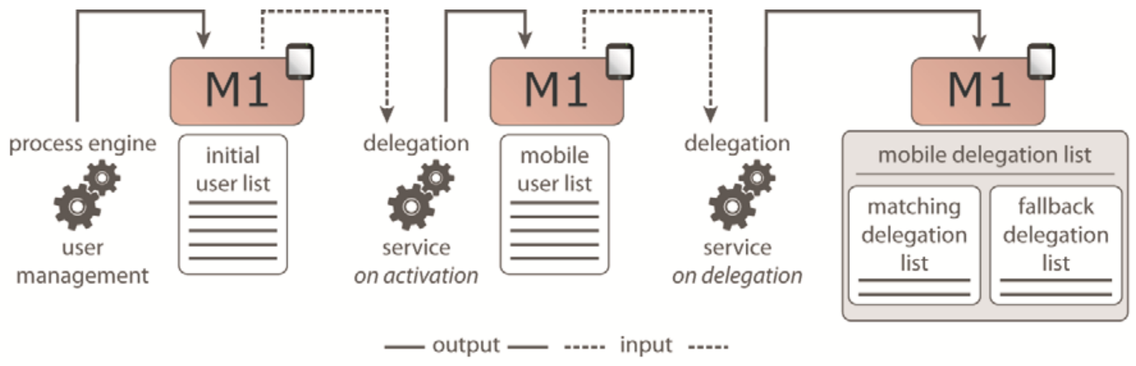


Figure 6. User list management

When a mobile task t_{mob} needs to be delegated, the *delegation service* first calculates dl_{mob}. Since this list shall allow for an automated delegation, it is sorted according to user priorities. The latter are determined based on the *battery status* of the users' devices as well as the number of the *instant shutdowns* they applied in the past (i.e., each instant shutdown of a user is recorded). Furthermore, dl_{mob} is split into two sub lists: a matching delegation list dl_{match}, which contains all users from ul_{mob} fulfilling the desired properties (i.e., connectivity, pre-filter, and location), and a fall-back delegation list dl_{fb} containing all users from ul_{mob} whose location differs from the one required by the task. Accordingly, users contained in dl_{match} are used as main target in the context of automated delegations. However, if no user from this list is available, users from dl_{fb} will be involved in future delegations as well. In particular, this approach uses the backup service at the latest possible time. In summary, the procedure for creating user list dl_{mob} works as follows:

```
$dl_{mob} \leftarrow (dl_{match}, dl_{fb})$
FOREACH $u_{mob}$ IN $ul_{mob}$
    $u_{mob}.priority \leftarrow (-1 * u_{mob}.instant.shutdown.counter)$
    IF $(u_{mob}.low.battery.status)$
        $u_{mob}.priority--$
    IF $(u_{mob}.connectivity)$ AND $(\neg u_{mob}.pre\text{-}filter)$
        IF $(u_{mob}.location = t_{mob}.location)$
            $dl_{match}.put(u_{mob})$
        ELSE
            $dl_{fb}.put(u_{mob})$
```

Altogether, using multiple user lists reduces the need for invoking the backup service during run time. Finally, the algorithm determining dl_{mob} uses the same weight for battery status and the shutdown counter to prioritize mobile users in dl_{mob}. To equally weight these two criteria is derived by the three mobile application scenarios discussed earlier in this chapter. In future work this will be further evaluated.

User Assignment and Race Conditions

The different lists maintained and presented above for the delegation and backup services prevent race conditions with respect to user assignments. Hence, only one mobile user can perform a mobile task during run time. To ensure this, the different lists are maintained as follows:

- The work lists maintained on the smart mobile devices to claim a mobile task are synchronized.
- Delegation management only prioritizes mobile users in ul_{mob} according to the presented aspects (e.g., battery status). As a result, the assignment of users is an atomic operation since it is performed similar to user assignments in existing business process engines (e.g., Dadam et al., 2009).

Adding Mobile Tasks to Process Execution

For adding a mobile task to a process model and hence for integrating it with process execution, two fundamental solutions exist. These will be presented in this section. In particular, it will be shown how the challenges summarized in Table 3 are addressed by these solution approaches:

Challenge	*Design Time*	*Instantiation Time*	*Activation Time*	*Delegation Time*
Connectivity	✗	✗	✓	✓
Low Battery	✗	✗	✓	✓
Instant Shut-Off	✗	✗	✓	✓
Location	✓	✗	✓	✓
UserLocation	✗	✗	✓	✓
Data Dependencies	✓	✗	✗	✗
Urgency	✓	✗	✗	✓
| (✓) : is evaluated | (✗) : is not evaluated |				

Table 3. Challenges for processes with mobile tasks

First, a backup service will be introduced, which changes the process model structure and adds a backup task to enable a robust execution of mobile tasks. Second, a delegation service will be defined that automatically delegates the execution of mobile tasks among available mobile users, if required. Before presenting these two services in detail, this section illustrates the basic steps required to add a mobile task to process execution. Overall, the procedure comprises four phases. Figure 7 shows in which of these phases manual (i.e., user interaction) and automated operations (i.e., delegation and backup service executions) are performed. Note that after creating an instance of a process model with mobile tasks, errors (e.g., connection loss of a smart mobile device) are handled automatically. This behavior can be ensured since delegation and backup services as well as the user list management are performed automatically.

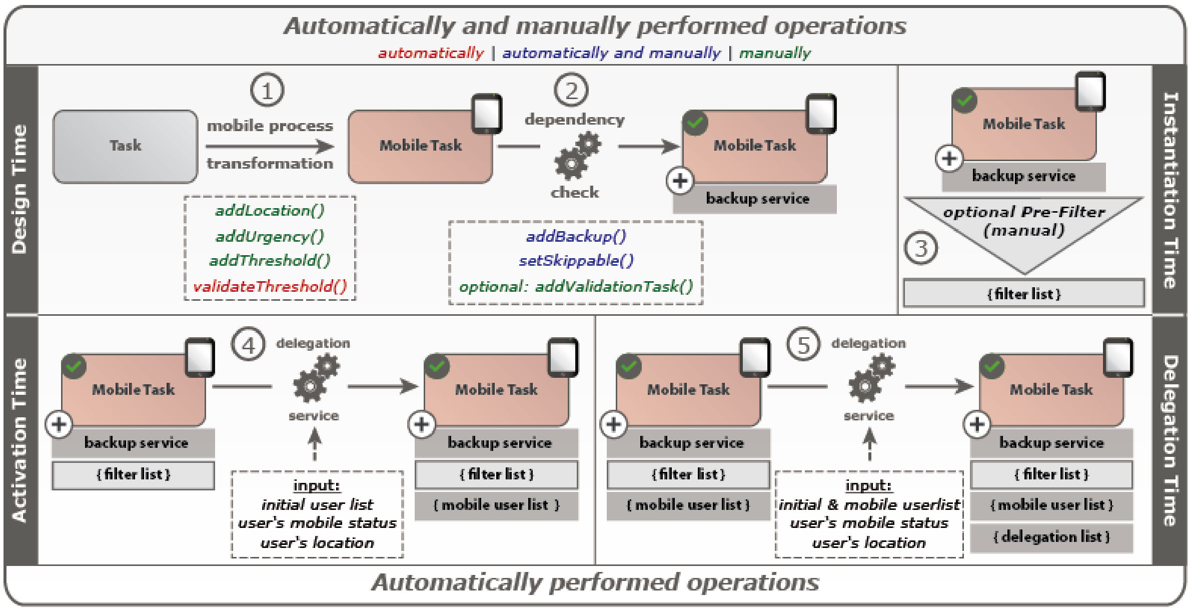


Figure 7. Procedure for integrating mobile tasks into process execution

Design Time

The design of a process model with mobile tasks consists of two phases. During the first one, which is called *mobile process transformation phase (see Figure 7(1))*, a process designer flags selected tasks of the given process model as *mobile*, i.e., these tasks shall be executed by mobile users on their respective smart mobile device during run time. In this context, the process designer may optionally assign a location, urgency, and threshold to each mobile task *(see Figure 7(1))*. The latter defines the minimum number of users that shall be available at run time in order to execute this task, i.e., the threshold allows controlling the delegation depending on the specific needs of the respective mobile process. For all mobile tasks, for which such a threshold is defined during this phase, the list of users who may perform this task is determined *(see Figure 7(1), validateThreshold)* based on information stored in the user repository. Finally, tasks whose threshold value lies beyond the number of currently available users are highlighted to the process designer who may then alter this value.

The second design time phase is the *dependency check (see Figure 7(2))*. In this phase, it is determined for which mobile tasks the backup service shall be provided. While the *mobile process transformation* is done manually (except the validation of the threshold), the *dependency check* can be performed automatically. First, all mobile tasks are analyzed with respect to the data elements they provide for subsequent process tasks. If a mobile task writes such data, the backup service will be added for this mobile task *(see Figure 7(2), addBackup)*. If this does not apply, the mobile task may be skipped during run time without need for additional exception handling, i.e., operation *setSkippable* may be applied to such a mobile task (*see Figure 7(2))*. Accordingly, attribute *IS_SKIPPABLE* of this mobile task is set to *true*. While the backup service (or *setSkippable* operation) is automatically added to a mobile task, the third operation of this phase (*see Figure 7(2)*, *addValidationTask)* is performed manually, i.e., the process designer must decide whether or not the execution of backup tasks must be manually confirmed during run time. In order to enforce this behavior, the process designer activates the validation tasks set by the backup service, i.e., the sync flag read by the validation tasks will be set to *true*.

Instantiation Time

When creating a process instance, a service is provided to change the run time configuration of this instance *(see Figure 7(3), addFilterList)*. This service shall cope with the dynamics of mobile environments. To perform such a change, the following steps are applied: First, for all mobile tasks, user lists are computed. Thereby, only currently online users are considered. Second, for each mobile task, it must be decided whether to change its location or urgency. In addition, users authorized to execute other *task*s may be removed. The latter option allows covering different kinds of mobile business scenarios properly. For example, the mobile device of a physician who needs to deal with an emergency should not be the target of upcoming mobile tasks.

During run time, the activation and delegation of mobile tasks are of particular interest for mobile task execution. Therefore, Figure 8 presents the lifecycle of a mobile task and relates these two particular run time phases with the entire run time lifecycle of mobile tasks (*see 3 & 4 in* Figure 7).

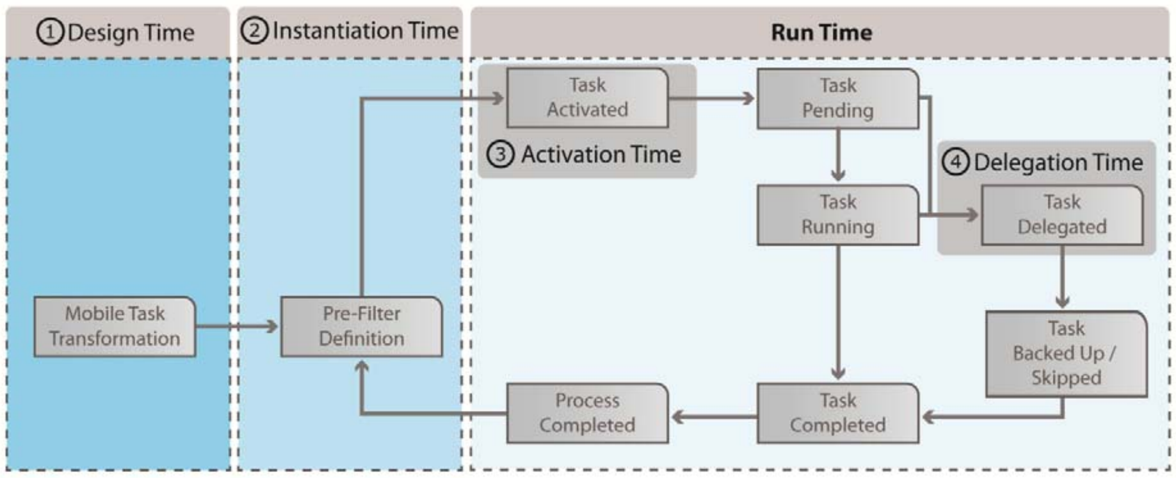


Figure 8. Mobile task lifecycle

Activation Time

When activating a mobile task, the following steps are performed automatically by the *delegation service (see Figure 7(4))*. First, the users who may perform the mobile task are determined *(see Figure 7(4), user list)*. A mobile user must meet the following criteria to be allowed to perform the mobile task *(see Figure 7(4), user's mobile status)*: First, a mobile user must be connected and the battery status of his mobile device must not be too low. Second, the mobile user should not have performed too many instant shutdowns previously. Third, if required, it will be automatically checked whether the mobile user is situated at the right location, i.e., whether attributes *UserLocation* and *Location* match (see Table 2).

Delegation Time

When delegating a mobile task at run time, it is automatically delegated to another user possessing the same rights. Further, for each mobile task, a delegation list is managed. This list will be created after the first delegation becomes necessary. It also stores a history of all delegations for this mobile task.

Generally, the following issues are crucial with respect to mobile task execution (see Figure 8):

- A failure of a mobile task, which produces data that is consumed by subsequent process steps, may cause severe run time errors (Reichert & Weber, 2012) (see *missing data in Figure 9*).
- An execution error of a mobile task might cause a deadlock (see *deadlock in Figure 9*), i.e., if a mobile task cannot be properly completed, succeeding tasks might not become activated.
- Regarding mobile task execution, usually, a time period is specified indicating the maximum duration of this task. For example, it might be required that a blood test is finished within 5 minutes (Lanz et al., 2013). Accordingly, any execution error of a mobile task should be handled in time in order to meet respective temporal constraints.

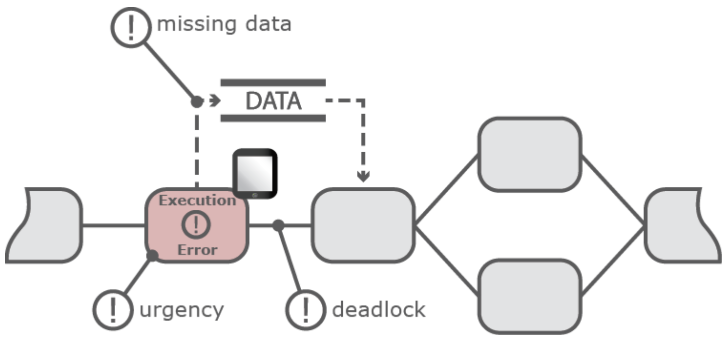


Figure 9. Mobile task execution challenges

SERVICE-ORIENTED SOLUTION APPROACH

The solution approach for the robust execution of mobile tasks, which is presented in this section, tackles the challenges discussed before. Table 4 gives an overview. For each service (i.e., mobile process transformation (*MPT*), backup service (*BS*), list management (*LM*), and mobile delegation service (*MDS*)) it shows which of these challenges it addresses.

Challenge	Component	Description
Connectivity	LM MDS	Only connected devices will be added to the user and delegation lists. The MDS refreshes these lists continuously and handles required delegations when a mobile device loses its connection.
Low Battery	LM	Only devices having a sufficient battery status will be added to the user and delegation lists.
Instant Shutdown	LM MDS	If a device is shut down, it will be removed from the user lists. Further, it will be re-added as soon as its status is *online* again. Moreover, the MDS only applies a delegation if the respective device will not be online for a longer period of time.
User Location	LM	If a location X is explicitly assigned to m*obile task*, only users whose current location matches X are added to the respective user lists.

Data Dependencies	BS	The backup service ensures that failures during the execution of a m*obile task* do not harm overall process execution, e.g., if subsequent tasks are data-dependent on the failed mobile task, the latter will be replaced by a respective backup task added to the process model.
Location	MPT	A m*obile task* may require a certain location for its execution.
Urgency	MPT MDS BS	The urgency of a task may be set at design time. In turn, the MDS then utilizes this information as trigger for delegating the mobile task, i.e., a backup service ensures that the mobile task will be always executed within the specified time frame.

Table 4. Challenges and solution components of mobile tasks

Delegation Service

During mobile task execution, the *mobile delegation service* (MDS) ensures that already assigned mobile tasks are automatically re-delegated to another authorized mobile user in case of errors. Since this delegation service maintains several user lists, the latter are first summarized before presenting the MDS.

User List Management

To enable a flexible delegation and hence to foster robust execution of a mobile task *t*, three user lists are maintained for t: ul_{init}, ul_{mob}, and dl_{mob}. User list ul_{init} contains all mobile users that are, in principle, authorized to perform mobile task *t*. Based on ul_{init} the mobile user list ul_{mob} is determined. Thereby, a mobile user from ul_{init} is only added to ul_{mob}, if the user is currently online, the user's location complies with the one of *t*, and the user is not excluded by any filter defined at instantiation time (see Figure 8). Based on ul_{mob}, *t is assigned to available* mobile users. Furthermore, if *t* shall be delegated, a mobile delegation list dl_{mob} is determined. First of all, all users contained in ul_{mob} are added to dl_{mob}. Then, dl_{mob} is ordered by taking the battery status, the number of instant shutdowns, and the locations of users into account. A low priority is assigned to a mobile user if the *battery status* of his mobile device is low or the respective *instant shutdown counter* is high. Both lists ul_{mob} and dl_{mob} are re-calculated when the connectivity status of a user from list ul_{init} changes.

Overall, when considering these three lists, the mobile delegation service may enter six different states (see Figure 10). The latter are denoted as *t(<STATE>)* and respective state transitions as T_i. Note that the delegation service starts when a mobile task *t* becomes activated. The delegation service will be activated before executing the respective task only if the mobile task has a value set for *urgency* and no authorized mobile user will promptly execute the task: Then, the latter will be delegated to an authorized mobile user, i.e., delegation is used for changing task state to running.

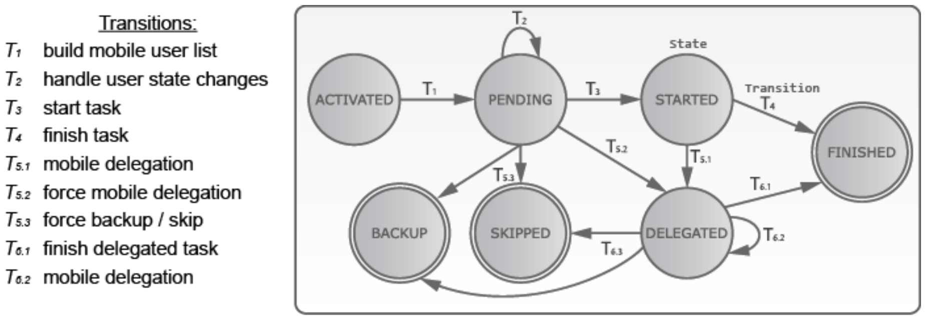


Figure 10. Mobile delegation service flow during run time

The scenarios shown in Table 5 are relevant when taking urgency[4] to_u ($to_u = 0$ denotes a timeout), user list threshold th_{mul}, and the ability to skip a mobile task t into account.

Scenario	Description	State Chain
Normal task execution	$user_a \in ul_{mob}$ starts mobile task t and performs it.	t(PENDING) → T3 → t(STARTED) → T4 → t(FINISHED)
Delegated task execution	$user_a \in ul_{mob}$ starts mobile task t. Then user state changes to offline or $to_u = 0$ holds, such that t will be automatically delegated to another user $user_b \in ul_{mob}$ who finishes the mobile task.	T3 → t(STARTED) → T5.1 → t(DELEGATED) → T6.1 → t(FINISHED)
Forced delegation	A forced delegation becomes necessary if the following holds: $t(PENDING) \wedge (\lvert ul_{mob} \rvert < th_{mul} \vee to_u = 0) \quad \vee$ $t(DELEGATED) \wedge (to_u = 0 \vee State(user_b)\ changes\ to\ offline) \Rightarrow$ t must be delegated to another user $user_n \in ul_{mob}$.	T5.2 → t(DELEGATED) ∨ T6.2 → t(DELEGATED)
Skip or Backup	Skip or backup will be performed if the following holds: If $(t(PENDING) \wedge to_u = 0 \wedge \lvert ul_{mob} \rvert = 0) \quad \vee$ $(t(DELEGATED) \wedge to_u = 0 \wedge \lvert dl_{mob} \rvert = 0)$. Furthermore, if IS_SKIPPABLE(T)=true, t will transit to SKIP, otherwise to BACKUP	t(PENDING) → T5.3 → t(SKIP) ∨ t(BACKUP)/ t(DELEGATED)→T6.3 → t(SKIP) ∨ t(BACKUP)

Table 5. Scenarios in which mobile delegation service is applied

Backup Service

A particular challenge arises if no mobile user is available for processing an activated mobile task that produces data during run time, i.e., if no delegation is possible anymore. In order to ensure that these mobile tasks can still be processed, a backup service is provided. Basically, it consists of two operations, which are added to a process fragment replacing the mobile task in case of exceptional situations. The first one is called *simple backup operation*, while the second one is denoted as *complex backup operation*. Before presenting the backup service in detail, the following issue is briefly discussed with respect to the backup service: How are skipping of mobile tasks and the backup service are related to each other? Regarding the latter question, consider Figure 11. The depicted backup service is only applied to mobile tasks that produce data. As shown in Figure 11, at best and delegation cases, a normal execution or delegation contributes to avoid the use of the backup service. Note that only in case no further delegation is possible (i.e., no more authorized mobile user for performing this task are available), the backup operation becomes necessary. Hence, the question can be answered as follows: Skipping a task and the backup service are not related.

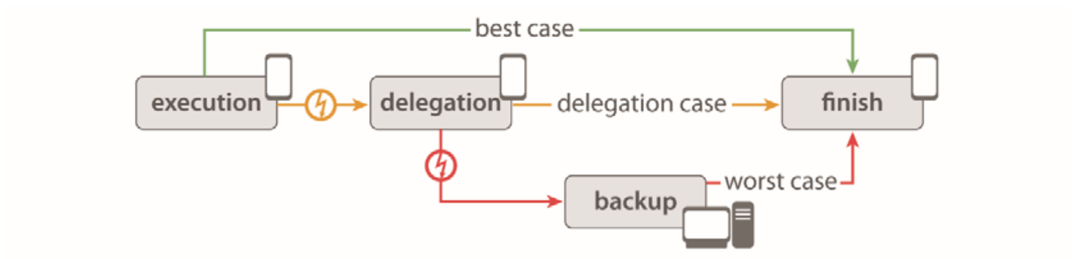


Figure 11. Use case of backup operation

[4] i.e., the specified period of time during which the task must be finished

In the following, the two backup operations as well as the context in which they are applied will be discussed.

Simple Backup Operation

During design time, all mobile tasks producing data for other tasks are determined. In turn, each of these tasks is then automatically equipped with a simple backup operation by applying the following steps: If a backup operation is needed for mobile task *B1*, the latter is substituted by the process fragment depicted in Figure 12. During run time, the execution of backup task *B2* on a stationary computer (see Figure 12) will then guarantee that subsequent tasks of *B1* will not be affected by any failure of this mobile task, i.e., backup task *B2* will provide the same data as mobile task *B1*.

In this context, a *sync flag* guarantees that B2 will be executed only if mobile task *B1* fails (see Figure 12). Thereby, B1 writes the *sync flag* according to its execution state. If *B1* has been executed correctly, the *sync flag* will be set to *true*, otherwise it will be set to *false*. Depending on the respective value, the succeeding XOR process fragment is then executed as follows: If the *sync flag* is "false", the upper branch will be chosen and *B2* be executed. In turn, if the *sync flag* is "true", the lower branch will be chosen and nothing happens; i.e., *B2* will only be executed if *B1* fails. As shown in Figure 12, the simple backup operation comprises another task, i.e., the *validation task*. The latter is used to manually confirm the execution of *B2*. If the *sync flag* is set to *true* and assigned to the *validation task* during design time, the following action will be performed during run time for confirming the task properly: The mobile user responsible for handling the failed mobile task must confirm that the backup task has completed correctly.

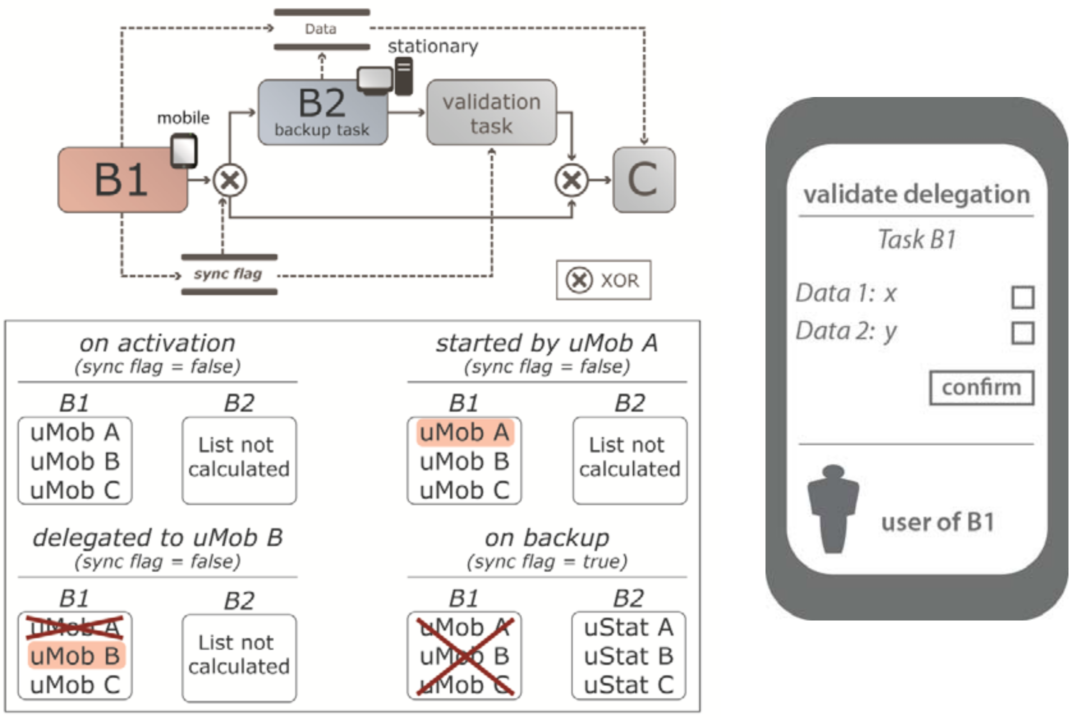


Figure 12. Simple backup operation and simplified validation task appearance during run time

Regarding the validation task of the backup operation, the following issue must be discussed: Does the validation task lead to an unfavorable behavior? Consider the following: If the validation task is not confirmed within a short period of time, the execution of the backup operation extends overall process execution time. In such case, for example, the backup operation may have the same bad impact as a

mobile user who is offline, which extends overall process execution time as well. In turn, in the MEDo project presented above, it turned out that exactly in this context most users demanded to confirm the execution of the mobile task if the backup operation has been applied instead. Furthermore, since the validation task is an optional task, the backup operation can be applied without using the validation task while ensuring the same robustness for overall process execution.
Finally, the following issue will be shortly discussed: Since the backup operation is performed on a stationary computer and hence failures are more unlikely than using a smart mobile device, the question arises, why trying to perform the respective task on a smart mobile device first? One can argue that in this case the mobile device is not necessary and affects overall robustness.
Regarding the latter question, recall that a mobile task shall be performed in a mobile manner. As a result, we want to provide mobile execution of a mobile task as long as authorized mobile users are available. Further recall that this behavior is provided by the delegation service. Then, only in case that no more mobile users are authorized that may execute the mobile task, the backup operation will be applied. As a result, on one hand, executing mobile tasks in a non-mobile manner if errors occur is actually not required in most cases. On the other, execution semantics will be preserved either when delegating the mobile task or applying the backup operation instead of it. For example, consider the following situation: Two ward physicians work on urgent mobile tasks with their mobile devices. Then, both devices run out of battery (frequently observed in the context of the above presented MEDo project). Further, no other physicians are able to perform these mobile tasks (frequently observed in the above mentioned MEDo project as well). In this case, using stationary computers to complete the tasks are highly welcome. Exactly, for this purpose, the backup operation will be used.

Complex Backup Operation

In order to deal with urgent mobile tasks, the complex backup operation depicted in Figure 13 is provided. It allows performing backup task *B2* more quickly based on two changes compared to the simple backup operation. First, a *user list task* is added. Second, the backup task is executed in parallel to the mobile task. In order to perform *B2* more quickly, the complex backup operations work as follows: First, the *user list task* determines the lists of authorized users for *B1* and *B2* respectively (see Figure 13, *on activation)*. Then, at the time *B1* is started, *B2* is started synchronously. Following this, *B2* will be not executable (but visible) for all users from the user list of *B2*. After assigning *B1* to a user (Figure 13, *started by uMob A*), the user list will be adapted for both tasks.

Note that the user list for *B2* assigns the task to the same user who has performed it on the smart mobile device as a mobile task. Applying this procedure offers advantages in several respects: First, all other users who may perform *B2* are able to monitor which mobile user is currently working on this task. Second, if for *B1* no more delegation to other authorized mobile users is possible, the user list for backup task *B2* has been already determined concurrently. Compared to the simple backup operation, for which the user list of *B2* is only determined when *B1 has been finished*, this procedure speeds up user assignment.

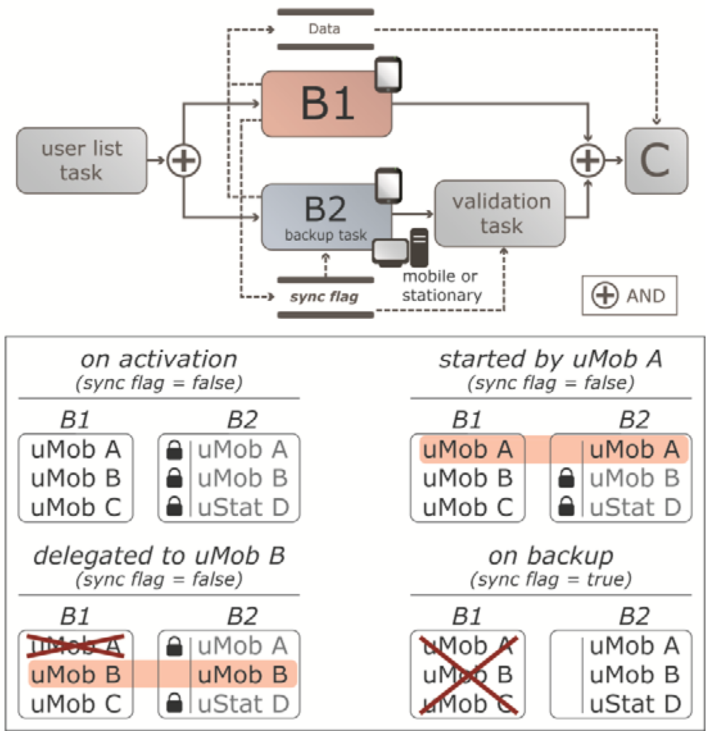


Figure 13. Complex backup operation

Figure 14 illustrates the use of both the simple and complex backup operation for a mobile task B.

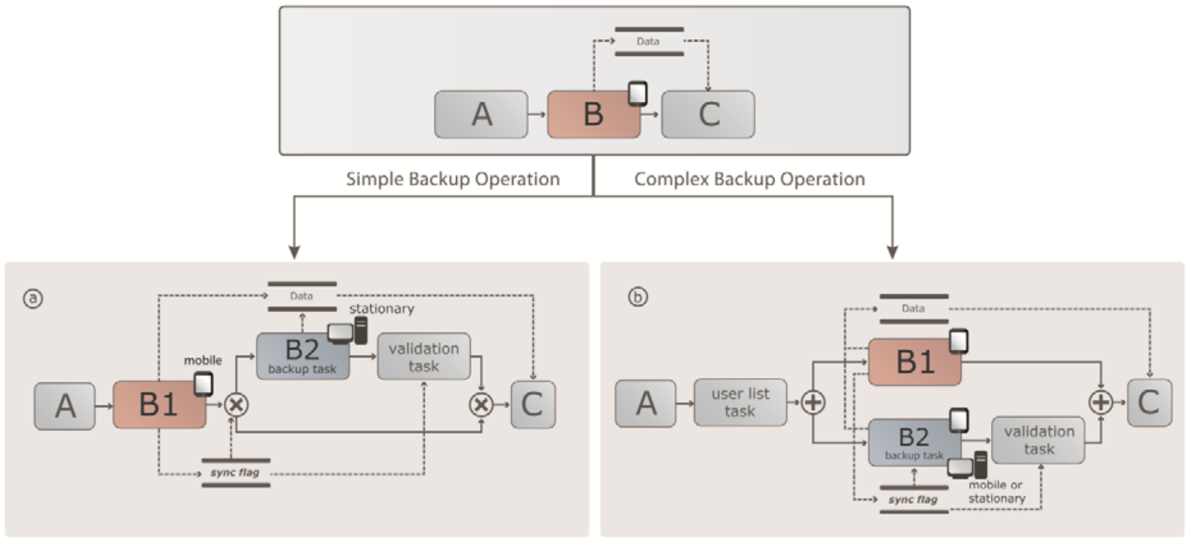


Figure 14. (a) Simple and (b) complex backup operation for mobile task B

Optimizing the Simple Backup Operation

This section discusses a potential optimization of the simple backup operation. This optimization was motivated by the fact that the use of a backup service might significantly extend the time required for executing a mobile task. This particularly applies to task sequences solely consisting of mobile tasks. An example of such a sequence is depicted in Figure 15 (i.e., process fragment C). Worst case, for all mobile tasks of this sequence (i.e., A1, B1, and C1), the backup service needs to be executed during run time. In turn, this is accompanied by several drawbacks: First, the overall execution time of A1, B1, and C1 is extended due to the additional time needed for executing the described backup and validation tasks. Second, the start of B1 is delayed when running the backup service of A1. The same applies to C1. Altogether, the backup service ensures proper run time behavior on one hand, but increases overall execution time on the other.

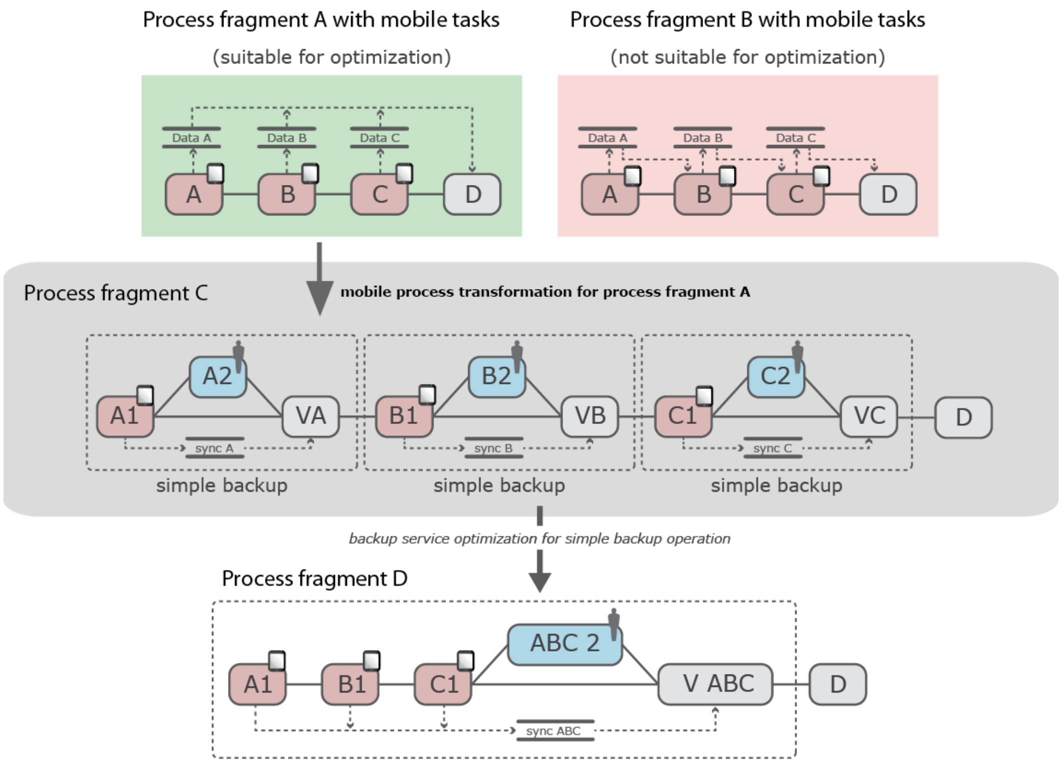

Process fragment A with mobile tasks
(suitable for optimization)
Process fragment B with mobile tasks
(not suitable for optimization)
Data A
Data B
Data C
A
B
C
D
Process fragment C
mobile process transformation for process fragment A
A1
A2
VA
sync A
B1
B2
VB
sync B
C1
C2
VC
sync C
simple backup
simple backup
simple backup
backup service optimization for simple backup operation
Process fragment D
ABC 2
V ABC
sync ABC

Figure 15. Optimization of simple backup operation

This section discusses a potential enhancement of the simple backup operation, which reduces the overall execution time required. The optimization is illustrated with respect to process fragment C as shown in Figure 15. It consists of a sequence of mobile tasks A1, B1, and C1 as well as their corresponding backup services. In the given scenario, none of the data elements written by the mobile tasks is read by any other mobile task of this sequence. In turn, this constitutes a prerequisite for applying the optimization described below. For example, this prerequisite is met by the process fragment depicted on the top left of Figure 15, while it is not satisfied by the process fragment shown on the top right.

The simple backup operation of process fragment C can be optimized as follows: Instead of applying the backup operation to all mobile tasks A1, B1, and C1, only one aggregated backup operation is provided for all of them. Furthermore, this aggregated backup service will then be applied after having executed the last mobile task, i.e., C1 (see process fragment D in Figure 15). Note that this aggregation is only applicable due to the fact that mobile tasks A1, B1, and C1 are not data-dependent on each other. If there had been data-dependencies between them, missing backup services of single tasks would harm further process execution. When applying this optimization to process fragment C, the benefit will be twofold: First, overall execution time is decreased since only one backup operation must be performed instead of three. Second, a mobile task succeeding another one for which the backup service shall be performed may be executed earlier since the required backup service is now applied afterwards. Finally, the aggregated backup service is applied like a *normal* backup service except for two aspects:

1. The aggregated backup task must provide missing data of more than one mobile task. This requires a user being allowed to write all these data elements. Consequently, the optimization is only possible if such a user exists; i.e., it is restricted to certain application scenarios.

2. If the validation tasks are set to true for all mobile tasks of the original task sequence, all mobile users must confirm the aggregated validation task. If less validation tasks are set to true, only these validation tasks must be confirmed.

IMPLEMENTATION

The sketched service-driven architectural approach (see Figure 4) has been implemented based on the AristaFlow process management technology (Dadam, et al., 2009; Reichert et al., 2009(b)). However, other process engines, like Intalio (Bhandari et al., 2011) or Activiti (Meister, 2011), meet the application programming interface requirements as well. Due to lack of space, we omit a detailed discussion of implementation issues. The MEE as well as *pm* and *md interfaces* (see Figure 4) are implemented in JAVA. The communication paradigm used for the two interfaces is based on REST Web services. Furthermore, only Android smart mobile devices can be used and integrated at the moment. In future implementations, we will add iOS and Windows mobile smart devices. It is noteworthy that the first prototype we implemented comprises all described services and functions, except the optimized backup operation (see above). In particular, the described delegation and backup services have been fully implemented. Finally, the functionality of smart mobile devices is restricted as follows: only simple user forms are provided on the smart mobile device for entering and changing task data. In future work, more sophisticated mobile task apps will be implemented (e.g., using sensors of the smart mobile device).

FUTURE RESEARCH DIRECTIONS

Amongst others, future research on mobile task support may consider the following issues:

- The delegation service should be enhanced in order to allow users to decline a delegation, if desired.

- In general, certain constraints may have to be obeyed when executing mobile tasks. For example, consider entailment constraints that may exist between different mobile tasks. When executing a mobile process, for example, it might be desirable that two mobile tasks are executed by the same user. Related research on integrating such constraints with business processes has received growing attention recently. However, realizing entailment constraints in the context of mobile processes and tasks raises additional issues, which must be integrated with our backup and delegation services.

- Assume that a mobile task has been delegated by a mobile user A to another mobile user B. Assume further that before the respective mobile task is finished by B, A recovers its smart mobile device. If A has already produced results conflicting with the ones of B, a conflict resolution is required.

- To provide aggregation for the backup service as presented by optimizing the simple backup operations is one possible approach of enhancing the entire backup service.

- A better location management is required for mobile tasks. For example, for a mobile *task a surrounding area* may be specified within which mobile users may execute mobile tasks.
- An approach to specify rules for enhancing the delegation service must be developed; e.g., users should be allowed to specify their own delegation rules.
- Finally, another area of interest for mobile task execution is cloud computing. Cloud spaces may be used to synchronize data written by smart mobile devices or to apply caching techniques more properly.

CONCLUSION

This chapter introduced an approach for enabling business processes with mobile task support. The presented backup service as well as mobile delegation service allow for a robust process execution. For this purpose, four fundamental issues need to be considered: First, the specific challenges of executing process tasks in a mobile environment must be well understood. Second, these challenges must be properly addressed at both design and run time. Third, mobile tasks must be executed in a robust way – the backup and mobile delegation services foster such robustness. Fourth, user acceptance is crucial in the context of mobile task support. Accordingly, the presented services do not involve mobile users in exception handling directly. Finally, a sophisticated architecture has been described showing how the presented approach has been realized in a service-oriented environment.

REFERENCES

[1]
van der Aalst, W. M. ; Weske, M.: The P2P Approach to Interorganizational Workflows. In Proc Advanced Information Systems Engineering (pp. 140-156), 2001.

[2]
Alonso, G. ; Agrawal, D. ; El Abbadi, A. ; Kamath, M. ; Gunthor, R. ; Mohan, C.: Advanced Transaction Models in Workflow Contexts. In Proc. of the 12th Int'l Conf on Data Engineering (pp. 574-581), 1996.

[3]
Alonso, G. ; Schek, H. J.: Research Issues in Large Workflow Management Systems. In Proc of NSF Workshop on Workflow and Process Automation in Information Science (pp. 126-132), 1996.

[4]
Alonso, G. ; Gunthor, R. ; Agrawal, D. ; El Abbadi, A. ; Kamath, M.: Exotica/FMDC: Handling Disconnected Clients in a Workflow Management System. In Proc 3rd Int'l Conf. on Cooperative Information Systems (pp. 99-110), 1995.

[5]
Ayora, C. ; Torres, V. ; Reichert, M. ; Weber, B. ; Pelechano, V.: Towards Run-Time Flexibility for Process Families: Open Issues and Research Challenges. In Proc Business Process Management Workshops (pp. 477-488), 2013.

[6]
Bhandari, R. ; Suman, U. ; Ramani, A. K.: Web Service Composition through BPEL Using Intalio. In Proc Computational Intelligence and Information Technology (pp. 873-876), 2011.

[7]
Baresi, L. ; Guinea, S.: Dynamo and Self-Healing BPEL Compositions. In Proc of the 29th Int'l Conf on Software Engineering (pp. 69-70), 2007.

[8]
Baresi, L. ; Guinea, S. ; Pasquale, L.: Self-Healing BPEL Processes with DYNAMO and the JBoss Rule Engine. In Proc Int'l Workshop on Engineering of Software Services for Pervasive Environments (pp. 11-20), 2007.

[9]
Baresi, L. ; Maurino, A. ; Modafferi, S.: Workflow Partitioning in Mobile Information Systems. In Proc of IFIP TC8 Working Conf on Mobile Information Systems MOBIS'04 (pp. 93-106), 2004.

[10]
Barros, A. ; Dumas, M. ; Oaks, P.: A Critical Overview of the Web Services Choreography Description Language. BPTrends Newsletter, 3, 1-24, 2005.

[11]
Battista, D. ; Leoni, M. ; Gaetanis, A. ; Mecella, M. ; Pezzullo, A. ; Russo, A. ; Saponaro, C.: ROME4EU: A Web Service-Based Process-Aware System for Smart Devices. In Proc ICSOC'08 (pp. 726-727), 2008.

[12]
Bauer, T. ; Reichert, M. ; Dadam, P.: Intra-Subnet Load Balancing in Distributed Workflow Management Systems. Int'l Journal of Cooperative Information Systems, 12, (pp. 205-223), 2003.

[13]
Cichocki, A. ; Rusinkiewicz, M.: Migrating Workflows. In Workflow Management Systems and Interoperability (pp. 339-355), 1998.

[14]
Crampton, J. ; Khambhammettu, H.: Delegation and Satisfiability in Workflow Systems. In Proc of the 13th ACM symposium on Access control models and technologies (pp. 31-40), 2008.

[15]
Crombach, A. ; Nandi, C. ; Bambonye, M. ; Liebrecht, M. ; Pryss, R. ; Reichert, M. ; Elbert, T. ; Weierstall, R.: Screening for Mental Disorders in Post-Conflict Regions using Computer Apps-a feasibility study from Burundi. In ESTSS Conference: Trauma and its clinical pathways: PTSD and beyond, SISST. Bologna, Italy, 2013.

[16]
Dadam, P. ; Reichert, M.: The ADEPT Project: a Decade of Research and Development for Robust and Flexible Process Support. Computer Science-Research and Development, 23(2), 81-97, 2009.

[17]
Dadam, P. ; Reichert, M. ; Rinderle-Ma, S. ; Lanz, A. ; Pryss, R. ; Predeschly, M. ; Kolb, J. ; Linh Thao, L. ; Jurisch, M. ; Kreher, U. ; Goeser, K.: From ADEPT to AristaFlow BPM Suite: a Research Vision has become Reality. In Business Process Management Workshops (pp. 529-531), 2009.

[18]
Gaaloul, K. ; Charoy, F.: Task Delegation Based Access Control Models for Workflow Systems. In Proc 9th IFIP Conf on e-Business, e-Services, and e-Society (pp. 400-414), 2009.

[19]
Geiger, P. ; Pryss, R. ; Schickler, M. ; Reichert, M.: Engineering an Advanced Location-Based Augmented Reality Engine for Smart Mobile Devices. Technical Report, University of Ulm, 2013.

[20]
Hackmann, G. ; Haitjema, M. ; Gill, C.: Sliver: A BPEL Workflow Process Execution Engine for Mobile Devices. In Proc ICSOC'06 (pp. 503-508), 2006.

[21]
Hahn, K. ; Schweppe, H.: Exploring Transactional Service Properties for Mobile Service Composition. In Proc MMS'09 (pp. 39-52), 2009.

[22]
Hallerbach, A. ; Bauer, T. ; Reichert, M.: Configuration and Management of Process Variants. In Int‘l Handbooks on Information Systems (pp. 237-255), 2010.

[23]
Isele, D. ; Ruf-Leuschner, M. ; Pryss, R. ; Schauer, M. ; Reichert, M. ; Schobel, J. ; Schindler, A. ; Elbert, T.: Detecting Adverse Childhood Experiences with a Little Help from Tablet Computers. In ESTSS Conference: Trauma and its clinical pathways: PTSD and beyond, SISST. Bologna, Italy, 2013.

[24]
Jones, V.M. ; van Halteren, A.T. ; Dokovski, N.T. ; Koprinkov, G.T. ; Peuscher, J. ; Bults, R.G.A. ; Konstantas, D. ; Widya, I.A. ; Herzog, R.: Mobihealth: Mobile Services for Health Professionals. Technical Report TR-CTIT-06-38, Enschede, 2006.

[25]
Knuplesch, D. ; Pryss, R. ; Reichert, M.: Data-Aware Interaction in Distributed and Collaborative Workflows: Modeling, Semantics, Correctness. In Proc 8th Int’l Conf on Collaborative Computing: Networking, Applications and Worksharing (pp. 223-232), 2012.

[26]
Knuplesch, D. ; Reichert, M. ; Pryss, R. ; Fdhila, W. ; Rinderle-Ma, S.: Ensuring Compliance of Distributed and Collaborative Workflows. In Proc 9th Int’l Conf on Collaborative Computing: Networking, Applications and Worksharing (pp. 133-142), 2013.

[27]
Kolb, J. ; Reichert, M.: A Flexible Approach for Abstracting and Personalizing Large Business Process Models. ACM Applied Computing Review, 13(1), (pp. 6-17), 2013.

[28]
Kunze, C.P.: DEMAC: A Distributed Environment for Mobility-Aware Computing. In Adjunct Proc of the 3rd Int’l Conf on Pervasive Computing (pp. 115-121), 2005.

[29]
Lanz, A. ; Weber, B. ; Reichert, M.: Time Patterns for Process-Aware Information Systems. Requirements Engineering, (pp. 1-29), 2013.

[30]
Lenz, R. ; Reichert, M.: IT Support for Healthcare Processes - Premises, Challenges, Perspectives. Data Knowledge Engineering, 61(1), (pp. 39–58), 2007.

[31]
Martin, D. ; Wutke, D. ; Leymann, F.: A Novel Approach to Decentralized Workflow Enactment. In Proc 12th Int'l IEEE Enterprise Distributed Object Computing Conference (pp. 127-136), 2008.

[32]
Meister, V.G.: Geschäftsregelbasierte Ansteuerung betrieblicher Anwendungssysteme am Beispiel der Open Source Process Engine Activiti. Betriebliche Anwendungssysteme, 65, 2011.

[33]
Philips, E. ; Van Der Straeten, R. ; Jonckers, V.: NOW: Orchestrating Services in a Nomadic Network using a Dedicated Workflow Language. Science of Computer Programming, 78(2), (pp. 168-194), 2013.

[34]
Pryss, R. ; Tiedeken, J. ; Reichert, M.: Managing Processes on Mobile Devices: The MARPLE Approach. In Proc CAiSE'10 Demos, 2010.

[35]
Pryss, R. ; Langer, D. ; Reichert, M. ; Hallerbach, A.: Mobile Task Management for Medical Ward Rounds - The MEDo Approach. In Proc BPM'12 Workshops (pp. 43–54), 2012.

[36]
Pryss, R. ; Mundbrod, N. ; Langer, D. ; Reichert, M.: Supporting Medical Ward Rounds Through Mobile Task and Process Management. Information Systems and e-Business Management, 2014.

[37]
Pryss, R. ; Musiol, S. ; Reichert, M.: Collaboration Support Through Mobile Processes and Entailment Constraints. In Proc 9th IEEE Int'l Conf on Collaborative Computing: Networking, Applications and Worksharing (pp. 178 - 187), 2013.

[38]
Pryss, R. ; Tiedeken, J. ; Kreher, U. ; Reichert, M.: Towards Flexible Process Support on Mobile Devices. In Proc CAiSE'10 Forum (pp. 150–165), 2010.

[39]
Reichert, M. ; Bauer, T.: Supporting Ad-Hoc Changes in Distributed Workflow Management Systems. In On the Move to Meaningful Internet Systems 2007: CoopIS, DOA, ODBASE, GADA, and IS (pp. 150-168), 2007.

[40]
Reichert, M. ; Weber, B.: Enabling Flexibility in Process-Aware Information Systems: Challenges, Methods, Technologies, 2012.

[41]
Reichert, M. ; Weber, B.: Process Change Patterns: Recent Research, Use Cases, Research Directions. In Seminal Contributions to Information Systems Engineering - 25 Years of CAiSE, (pp. 398–404), 2013.

[42]
Reichert, M. ; Bauer, T. ; Dadam, P.: Flexibility for Distributed Workflows. In Handbook of Research on Complex Dynamic Process Management (pp. 137–171), IGI Global, 2009.

[43]
Reichert, M. ; Dadam, P. ; Rinderle-Ma, S. ; Lanz, A. ; Pryss, R. ; Predeschly, M. ; Kolb, J. ; Thao Ly L. ; Jurisch, M. ; Kreher, U. ; Goeser, K.: Enabling Poka-Yoke Workflows with the AristaFlow BPM Suite. In Proc BPM'09 Demonstration Track, 2009.

[44]
Ruf-Leuschner, M. ; Pryss, R. ; Liebrecht, M. ; Schobel, J. ; Spyridou, A. ; Reichert, M. ; Schauer, M.: Preventing Further Trauma: KINDEX Mum Screen-Assessing and Reacting Towards Psychosocial Risk Factors in Pregnant Women with the Help of Smartphone Technologies. In ESTSS Conference: Trauma and its clinical pathways: PTSD and beyond, SISST. Bologna, Italy, 2013.

[45]
Schaad, A.: A Framework for Organisational Control Principles. PhD thesis, The University of York, England, 2003.

[46]
Schmidt, H. ; Hauck, F.J.: SAMPROC: Middleware for Self-Adaptive Mobile Processes in Heterogeneous Ubiquitous Environments. In Proc 4th Middleware Doctoral Symposium (pp. 1-6), 2007.

[47]
Schmidt, H. ; Kapitza, R. ; Hauck, F.J.: Mobile-Process-Based Ubiquitous Computing Platform: A Blueprint. In Proc 1st Workshop on Middleware-application interaction (pp. 25-30), 2007.

[48]
Schobel, J. ; Schickler, M. ; Pryss, R. ; Nienhaus, H. ; Reichert, M.: Using Vital Sensors in Mobile Healthcare Business Applications: Challenges, Examples, Lessons Learned. In Proc 9 Int'l Conf on Web Information Systems and Technologies (pp. 509–518), 2013.

[49]
Stürmer, G. ; Mangler, J. ; Schikuta, E.: Building a Modular Service Oriented Workflow Engine. In Proc Service-Oriented Computing and Applications (pp. 1-4), 2009.

[50]
Tuysuz, G. ; Avenoglu, B. ; Eren, P. E.: A Workflow-Based Mobile Guidance Framework for Managing Personal Activities. In Proc 7th Int'l Conf on Next Generation Mobile Apps, Services and Technologies (pp. 13-18), 2013.

[51]
Wakholi, P. K. ; Chen, W.: Workflow Partitioning for Offline Distributed Execution on Mobile Devices. In Proc of the CAiSE'12 Forum at the 24th Int'l Conf on Advanced Information Systems Engineering (pp 171-178), 2012.

[52]
Weber, B. ; Reichert, M. ; Rinderle-Ma, S.: Change Patterns and Change Support Features - Enhancing Flexibility in Process-Aware Information Systems. Data Knowledge Engineering, 66(3), (pp. 438–466), 2008.

[53]
Wodtke, D. ; Weikum, G.: A Formal Foundation for Distributed Workflow Execution based on State Charts. In Database Theory—ICDT'97 (pp. 230-246), 1997.

[54]
Zaplata, S. ; Dreiling, V. ; Lamersdorf, W.: Realizing Mobile Web Services for Dynamic Applications. In Proc I3E'09 (pp. 240-254), 2009.

[55]
Zaplata, S. ; Hamann, K. ; Kottke, K. ; Lamersdorf, W.: Flexible Execution of Distributed Business Processes Based on Process Instance Migration. Journal of Systems Integration, 1(3), (pp. 3-16), 2010.

[56]
Zaplata, S. ; Kottke, K. ; Meiners, M. ; Lamersdorf, W.: Towards Run Time Migration of WS-BPEL Processes. In Proc WESOA'09 (pp. 477-487), 2010.