Collaboration Support Through Mobile Processes and Entailment Constraints

Rüdiger Pryss
Institute of Databases and Information Systems
University of Ulm, Germany
Email: ruediger.pryss@uni-ulm.de

Steffen Musiol
Institute of Databases and Information Systems
University of Ulm, Germany
Email: steffen.musiol@uni-ulm.de

Manfred Reichert
Institute of Databases and Information Systems
University of Ulm, Germany
Email: manfred.reichert@uni-ulm.de

***Abstract*—The computational capability of smart mobile devices increasingly fosters their prevalence in many business domains. Along this trend, process management technology is going to be enhanced with mobile task support. However, tasks executed stationarily so far cannot be simply transfered to mobile devices. For the latter purpose, we developed an approach within the MARPLE project enabling mobile and robust task execution in the context of business processes. In particular, this approach provides self-healing techniques that relieve mobile users from manually handling errors (e.g., lost connections) during mobile task execution. In this paper, we extend the collaboration facilities of our approach by adding entailment constraints to mobile task management. In the context of a business process, for example, two tasks may have to be executed by the same (mobile) 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 probably integrated with the mentioned error handling techniques. We present fundamental entailment constraints supported by our approach and discuss how they can be realized in a robust and flexible manner. In particular, this will significantly enhance mobile task and process support in next generation information systems.**

***Keywords*—mobile process, flexibility, mobile task support, entailment constraints**

I. Introduction

Knowledge workers increasingly demand for a mobile and flexible access to information systems. However, the integration of mobile devices into an existing IT infrastructure is laborious and error-prone. In particular, the infrastructure must cope with ad hoc events, errors (e.g. connectivity problems), physical limitations of mobile devices (e.g., limited battery capacity), unexpected user behaviour (e.g., instant shutdowns), and mobile data collection [1].

In general, proper exception handling is crucial for enabling mobile process and task support. In this context, flexible process management technology offers promising perspectives based on a wide range of techniques (e.g., ad hoc changes) [2]–[6]. In particular, these techniques foster robust mobile task support during process execution as well. However, executing tasks on mobile devices in the same way as on stationary computers is usually not appropriate. For example, consider the execution time required for processing a task. While for tasks executed on stationary computers a long duration usually has no particular effect, this is no longer the case for tasks running on mobile devices, for which exceptions like a broken device or lost connection might occur. Hence, any approach supporting mobile tasks must consider the specific challenges of a mobile environment (see [7], [8] for case studies we conducted in this context).

In previous work within the MARPLE project, we developed an approach that enables a robust execution of mobile tasks in the context of business processes. The various challenges we have addressed in this context are related to the following three categories: (1) challenges raised by the mobile environment itself (e.g., lost connections of mobile devices), (2) challenges related to business process execution (e.g., missing process data due to task failures), and (3) challenges caused by misbehaviour of the mobile user (e.g., mindless instant shutdowns). Our approach provides two fundamental concepts to deal with these challenges, i.e., a *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. To avoid wrong or unfavorable delegations in this context, the service considers the context of mobile users as well. Finally, if no suitable mobile user can be determined, the backup service ensures that overall process execution will not be harmed.

In this context, a particular challenge is raised by the need to satisfy a number of *entailment constraints*. As examples of such constraints consider *separation of duties* (i.e., two particular tasks of a process instance must be executed by *different* users) and *binding of duties* (i.e., two particular tasks of a process instance must be executed by the *same* user). In general, an entailment constraint defines a dependency between tasks [9]. Regarding mobile process and task support, respective constraints are crucial in order to ensure proper collaboration among actors. However, satisfying respective constraints in a mobile process context is a challenging task to accomplish [9].

In a number of case studies, we identified three kinds of entailment constraints being of particular interest: binding of duties, separation of duties, and cardinality. The latter is a constraint defined on a multi-instance task and restricts the number of its executions (i.e., instances of this task at runtime). When integrating entailment constraints in our approach enabling mobile process support additional challenges arise. In particular, we must enhance the aforementioned mobile delegation service properly. This paper shows how we have accomplished this approach. In particular, we show how to

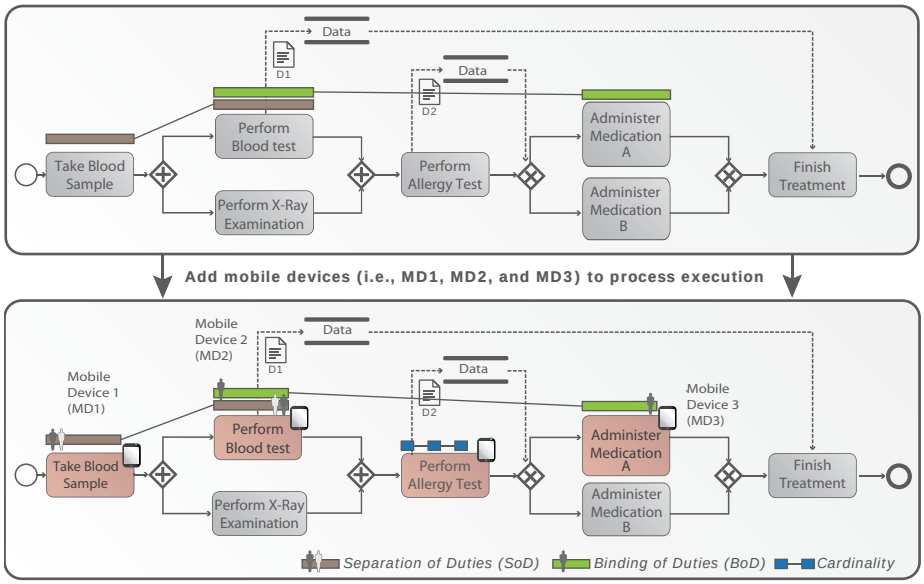


Figure 1. Adding mobile devices to process execution (i.e., the activities colored in red)

avoid burdening mobile users with manual tasks in case of errors.

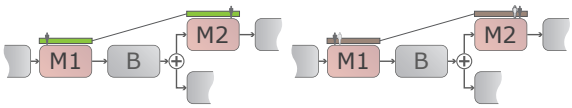
The remainder of this paper is organized as follows: Section II describes the entailment constraints we consider, discusses relevant challenges, and summarizes the major contributions of our work. Section III presents background information on our approach enabling mobile process and task support and discusses the specific challenges to be addressed. Further, we describe the steps to be performed during design and runtime in order to provide mobile task support in the context of business processes. Finally, we present two fundamental concepts supporting the robust execution of mobile tasks, i.e., the backup and delegation service for mobile tasks. Section IV shows how to add entailment constraints to the mobile delegation service. Section V then presents details regarding the implementation of our core approach and the way it integrates entailment constraints. Finally, Section VI discusses related work and Section VII concludes with a summary and outlook.

II. Entailment Constraints: Example and Challenges

First, we sketch the entailment constraints we consider in this paper. Second, we present a healthcare scenario to illustrate the challenges that emerge when integrating these constraints with mobile process and mobile task support. In the context of our work, entailment constraints are considered with respect to business process execution (or to be more precise, the execution of process instances according to a pre-specified process model). For example, ensuring that two tasks are not executed by the same user is required in many business scenarios, e.g., a credit application must not be approved by the same person who requested the credit.

A. Entailment Constraints

Figures 2 and 3 illustrate how we annotate a process model with entailment constraints. In Fig. 2(a), denoted by the connected green rectangles placed on top of mobile tasks



(a) Binding of Duties (b) Separation of Duties

Figure 2. Binding and separation of duties

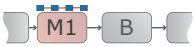


Figure 3. Cardinality

M_1 and M_2, a binding of duties is defined. According to this specification for any process instance, the mobile user performing M_1 must perform M_2 as well. In a healthcare context, for example, it must be ensured that the physician who examines the patient also administers her medication. In turn, in Fig. 2(b), denoted by the connected brown rectangles placed on top of mobile tasks M_1 and M_2, a separation of duties is defined. It expresses that a mobile user performing M_1 must not perform M_2. For example, in the context of a credit request, the person launching a credit enquiry must not approve it.

Finally, in Fig. 3, denoted by inter-connected blue rectangles placed on top of mobile task M_1, a cardinality constraint is defined. It expresses that multi-instance task M_1 will be executed a pre-specified number of times. Regarding the integration of these constraints into our approach, we make the following assumptions:

- Separation as well as binding of duties define a dependency between mobile tasks M_1 and M_2, with M_1 preceding M_2, i.e., $M_1 < M_2$. We neither consider separation of duties nor binding of duties for mobile tasks belonging to parallel branches. Further, we exclude respective constraints for mobile tasks surrounded by a loop structure at this stage. The latter applies to the cardinality constraint as well.
- A mobile task may be referred by more than one entailment constraint. For example, a separation of duties between mobile tasks M_1 and M_2 may be accompanied by a binding

of duties between M_2 and M_3.

B. Challenges

To illustrate the challenges that emerge when integrating the three kinds of entailment constraints with mobile process and mobile task support, we consider the healthcare scenario from Fig. 1. For example, between mobile tasks *Take Blood Sample* and *Perform Blood Test*, a separation of duties constraint is defined. Assume that at the time task *Take Blood Sample* shall be executed, three authorized mobile users (i.e., Miller, Mayer, and Neuhann) are available. In addition, the same users are authorized to perform task *Perform Blood Test* later on. Furthermore, the mobile device of user Miller, who is actually performing the task *Perform Blood Test*, encounters physical problems. In this case, the process cannot terminate properly, since task *Finish Treatment* is data-dependent on this mobile task. In this scenario, the aforementioned mobile delegation service will automatically delegate task *Perform Blood Test* to another authorized mobile user available (e.g., Mayer).
Furthermore, we must consider the separation of duties between tasks *Take Blood Sample* and *Perform Blood Test* as well. Since two mobile users have already worked on this task, only mobile user Neuhann will be allowed to execute *Perform Blood Test*. Consequently, for task *Perform Blood Test*, no delegation is possible. Note that this scenario might raise additional problems. Since *Perform Blood Test* requires a binding of duties with *Administer Medication A*, for example, only user Neuhann will be allowed to execute the latter task (i.e., *Administer Medication A*) in our scenario. Overall, this simple scenario shows that any mobile delegation service might affect the enforcement of entailment constraints.

III. Mobile Process and Task Support

This section presents basic concepts of our approach enabling mobile process and task support. First of all, we discuss the challenges addressed by it. Then, we present a procedure for integrating *mobile tasks* into business process execution. In this context, we show in which phases constraints may be added and evaluated. Finally, we present the core components of our approach, i.e., its backup and mobile delegation services.

A. Challenges in mobile environments

To enable a robust integration of mobile devices into process execution, the fundamental challenges of mobile environments need to be addressed. In particular, the state of mobile devices as well as the behaviour of mobile users must be taken into account, since both might harm overall process execution. In addition, we must consider the specific challenges related to the execution of a mobile process, i.e., a process for which a subset of its tasks is executed on mobile devices. In the following, we categorize these challenges into process-, environment-, and user-related ones.

Connectivity ***(environment)***. Connectivity refers to the availability of users and the mobile devices assigned to them. Hence, unavailability might be due to the status of a device (e.g., broken device) or a personal status (e.g., user is on vacation). If a mobile device is connected to a network, it will be used as a potential target device for executing *mobile tasks*, otherwise it will be not.

Low Battery ***(environment)***. A device already indicating a low battery status should not be the target platform for executing an upcoming *mobile task* until the battery is recharged; i.e., a low battery status means that we do not consider this mobile device (and its user) at the moment. Furthermore, that users have backup devices is not considered at this stage.

Instant Shutdown ***(user behaviour)***. In practice, it happens that users instantly shut down their mobile device without reflecting on the consequences of this shutdown. Usually, this constitutes a short-term problem and the device can be restarted soon in most cases. If a user exhibits many instant shutdowns, however, this misbehaviour will be considered in our approach. To deal with such "careless" shutdowns, a mobile device sends a message to our services indicating that an instant shutdown will take place soon. In this context, we evaluated several mobile development frameworks (i.e., *Google Android, Apple iOS, Microsoft Windows Mobile*) and were able to demonstrate that we can apply this solution for detecting instant shutdowns to all of them. Finally, to assess user behaviour over time (e.g., whether or not she performs many instant shutdowns), our approach manages the number of instant shutdowns applied.

User Location ***(user behaviour)***. At runtime, for mobile users, attribute *UserLocation* indicates where these users are located. If a mobile user shall execute a *mobile task* at a location different from her present one (e.g., if her coordinates differ from the ones defined for the task), this will be considered.

Data Dependencies ***(process)***. Data dependencies between process activities can be derived from the order in which activities read and write data objects. In our approach, we explicitly consider *mobile tasks* with data dependencies.

Location ***(process)***. A *mobile task* has an attribute *Location* that optionally stores the location this task shall be performed. Note that in certain cases, data or physical objects needed to accomplish a task, are only available at a certain location. If the user is performing her work, while continuously moving, it cannot be guaranteed that she is on the right spot to gather data needed. For example, if a physical examination of a patient is assigned to a physician who continuously switches her location within the hospital, neither gathering patient data nor performing the examination will be meaningful tasks for this physician at the present moment.

Urgency ***(process)***: This task attribute reflects the urgency of a *mobile task*. For example, if a lab test is required in the context of an emergency surgery, urgency of a task performing this lab test will be high. The value of this attribute either is *null* or describes the point in time the *mobile task* shall be performed, i.e., either a concrete point in time or a period. If a period is specified, after allocating the *mobile task* to a mobile user, she must finish it within the specified period.

B. Executing Processes with Mobile Tasks

We introduce the way we use mobile devices for executing *mobile tasks* in the context of a business process. Understanding this is crucial for realizing an approach that integrates

mobile devices with business process execution. Basically, two options exist: First, a mobile device may be used as process client to which tasks may be assigned at runtime. In this case, the mobile device covers a subset of the functionality of a stationary process client. In particular, it comprises a worklist that will be continuously updated by the process engine. Second, the mobile device itself might run a local process engine and be able to autonomously execute an entire process or process fragment. In [10] and [11], we have given insights into the latter approach, whereas this paper focuses on the integration of mobile devices following the first approach. Both approaches are part of our MARPLE project focusing on a tight integration of process management technology and mobile computing.

C. Adding Mobile Tasks to Process Execution

This section introduces the four phases for integrating a *mobile task* into process execution. Thereby, we illustrate how our overall integration procedure works (cf. Fig. 5). Amongst others, Fig. 5 indicates in which phases a manual interaction (i.e., user interaction) and in which an automatic processing (i.e., automated service operations) become possible. Furthermore, in Fig. 4 we illustrate in which phases the mentioned challenges are considered. Thereby, Fig. 4 also shows in which phases the entailment constraints are considered.

Regarding our procedure for adding *mobile tasks* to process execution (cf. Fig. 5), we suggest two fundamental concepts addressing the challenges illustrated in Fig. 4. First, we define a *backup service*, which changes the process structure by adding a backup task executed on a computer. This ensures robust execution of *mobile tasks*. Further, note that in certain cases even a backup task may be executed on a mobile device. Second, we define a delegation service that automatically delegates the execution of *mobile tasks* to other authorized mobile users in case of failures (i.e., non-availability of a particular mobile task). In addition, this service handles issues related to the entailment constraints. These two techniques are presented in Sections III(D+E), while this section illustrates the context in which they are applied.

$Design\ Time\ Phase$. Design time composes two phases. The first one is called *mobile process transformation* ①. During this phase, a process designer may declare arbitrary tasks as mobile, which means that they shall be executed on a mobile device. Following this, he may optionally assign a location, an urgency, and a threshold to these *mobile tasks* (cf. ①). Regarding entailment constraints, we must distinguish two cases: First, an entailment constraint may have been already defined for a process task that is now transformed into a mobile one. Second, no constraint has been defined so far, but shall be added to the *mobile task*. In this case, a process designer must specify the respective constraint. Further, for each mobile task its threshold defines the minimum number of users that should be available at runtime to execute this task; i.e., the threshold allows us to control delegation according to particular needs of the (mobile) process. In addition, for all *mobile tasks* associated with a threshold, during this phase, we compute the list of users that may perform the task (cf. ①, $validateThreshold$). For this purpose, we access the user repository. Tasks, for which the threshold is set to a value beyond the number of available users, are highlighted to the process designer who may then alter the threshold. Note that threshold validation is performed automatically. The second design time phase we consider is the *dependency check* ②. In this phase, we determine for which *mobile tasks* the backup service shall be performed. While the *mobile process transformation* is accomplished manually (except the validation of the threshold), the *dependency check* is performed automatically. First, all *mobile tasks* are analyzed according to their data dependencies with other process tasks. If a *mobile task* writes data for subsequent tasks, the backup service will be added for this *mobile task* (cf. ②, $addBackup$). In case no data dependency exists for a *mobile task*, it may be skipped during runtime; i.e., operation $setSkippable$ may be performed on this *mobile task* (cf. ②), i.e., attribute *IS_SKIPPABLE* of this *mobile task* is set to *true*. While the first two options are performed automatically, the last action of this phase (cf. ②, $addValidationTask$) is performed manually. Thereby, a process designer must decide how the validation tasks of the backup service shall be evaluated during runtime. We present the semantics of the validation task in the context of the backup service in Section III(B).

Aspects of Mobility	Process Design Time	Process Instantiation Time	Task Activation Time	Task Delegation Time
Connectivity	✗	✗	✓	✓
Low Battery	✗	✗	✓	✓
Instant Shut-Off	✗	✗	✓	✓
Location	✓	✗	✓	✓
UserLocation	✗	✗	✓	✓
Data Dependencies	✓	✗	✗	✗
Urgency	✓	✗	✗	✓
Entailment Constraints	✓	✗	✓	✓
| (✓) : is evaluated | (✗) : is not evaluated |				

Figure 4. Challenges of processes with *mobile tasks*

$Instantiation\ Phase$. When creating a process instance, we provide a service to change the runtime configuration for this instance (cf. ③, $addFilterList$). This enables us to cope with the dynamics of mobile environments more properly. To perform such change, the following steps are made. First, for all *mobile tasks* we compute user lists. Thereby, we only consider users who are currently online. Second, for each *mobile task*, it must be decided whether to change its location, urgency, or entailment constraints. In addition, users authorized to execute this *mobile task* may be removed. The latter option enables us to cover mobile business scenarios more properly. For example, the mobile device of a physician who deals with an emergency, should not be the target for *mobile tasks*.

$Activation\ Phase$. When activating a *mobile task*, the following steps are performed automatically by the *delegation service* (cf. ④). First, the users who may perform the *mobile task* are determined (cf. ④, $user\ list$). In this context, a mobile user must meet the following to be allowed to perform the *mobile task* (cf. ④, $user's\ mobile\ status$): First, she must be connected and the battery status of her mobile device must not be low. Second, she should not have performed too many instant shutdowns. Third, if relevant, it will be (automatically) checked whether the mobile user is at the right location, i.e., whether attributes *UserLocation* and *Location* match (cf. Fig. 4). Finally, constraint dependencies are considered. Note that the latter might require a specific handling of the user list of a *mobile task*.

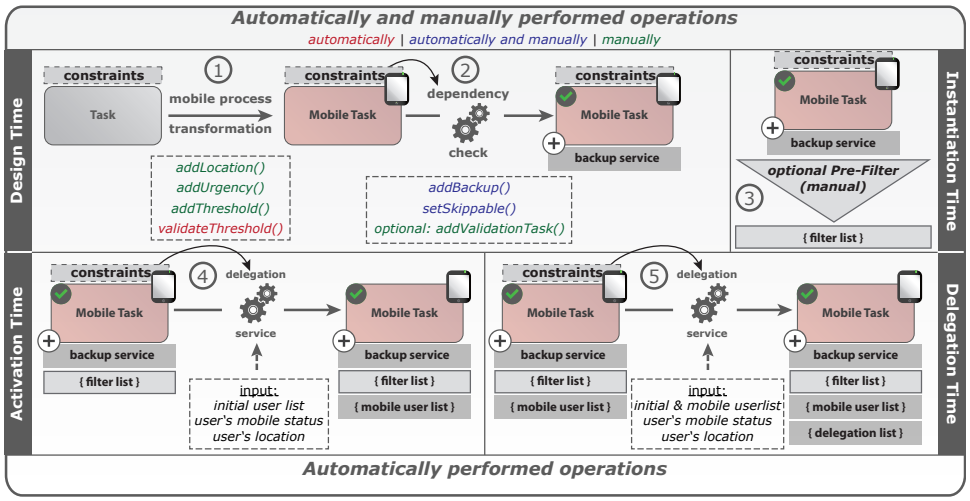


Figure 5. Procedure of integrating *mobile tasks* into process execution

Delegation Phase. When delegating a *mobile task* at runtime, we add a delegation list to its *mobile task* in order to be able to track to which mobile users it has been delegated.

D. Backup Service

When combining business processes with *mobile task* support, a particular challenge emerges if no mobile users are available for an activated *mobile task* at runtime (see Section III(A) for reasons causing this situation). In order to ensure that *mobile tasks* will still be executed in such a scenario, our approach provides a backup service, which comprises two operations. In particular, these operations are embedded in a process fragment that replaces the *mobile task* if the aforementioned problems occur at runtime. Fig. 6 shows an example of such a fragment. Note that the two backup operations have two preconditions. First, the operations may only be applied to *mobile tasks*. Second, a *mobile task* must provide data for subsequent tasks. In particular, such *mobile tasks* may harm overall process execution. To be more precise, if the tasks succeeding a *mobile task* M in the flow of control consume data provided by M, a deadlock or other error might occur in case a failure of M is not handled properly. To avoid such situations, we provide the backup service described in the following. First of all, we present the simple backup operation. Following this, we present the complex backup operation and discuss its difference to the simple one.

During design time, we identify all *mobile tasks* producing data for other tasks. Following this, we automatically apply the simple backup operation to all these tasks. Thereby, the following steps are applied: If a backup operation is needed for a *mobile task* B1, it will be substituted by the process fragment depicted in Fig. 6 at design time. During runtime, the execution of backup task B2 on a stationary computer will then guarantee that subsequent tasks of B1 will not be affected by a failure of this *mobile task*, i.e., the backup task B2 will provide the same data as *mobile task* B1. In this context, the *sync flag* guarantees that B2 will be only performed if *mobile task* B1 fails (cf. Fig. 6). Thereby, B1 writes the *sync flag* according to its execution state. If B1 has been executed correctly, the *sync flag* is set to $true$, otherwise it will be set to $false$. Depending on the value of the *sync flag*, the subsequent XOR process fragment is then executed as follows: If the *sync flag* is $true$, the upper branch will be chosen and B2 be executed. In turn, if the *sync flag* is $false$ the other branch will be chosen and nothing more happens. Therefore, B2 will only be executed if B1 fails.

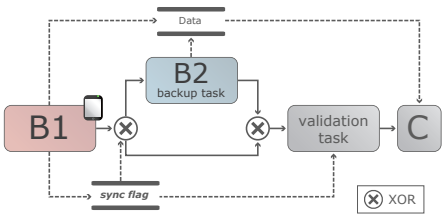


Figure 6. Simple backup operation

As shown in Fig. 6, the simple backup operation comprises another task, i.e., the *validation task*. We use the validation task to manually confirm the execution of B2. If, at design time, the *sync flag* is set to $true$ and assigned to the *validation task*, the mobile user responsible for handling the failed *mobile task* must confirm at runtime that the backup task has been completed correctly.

In order to deal with urgent *mobile tasks*, we provide the complex backup operation shown in Fig. 7. It allows performing the backup task B2 more quickly. We changed two aspects compared to the simple backup operation. First, we add a *user list task*. Second, the backup task is executed in parallel to the *mobile task*. In order to perform B2 more quickly, the complex backup operations works as follows: First, the user list task determines the lists of authorized users for tasks B1 and B2 (cf. Fig. 7, *on activation)*. Then, at the time B1 is started, B2 is started synchronously. Following this, the task will be locked for all users from the user list of B2. After assigning B1 to a user (cf. Fig. 7, *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 mobile device as a *mobile task*. Applying this procedure is advantageous 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 authorized mobile users is possible, we have already determined the user list for backup task B2. Compared

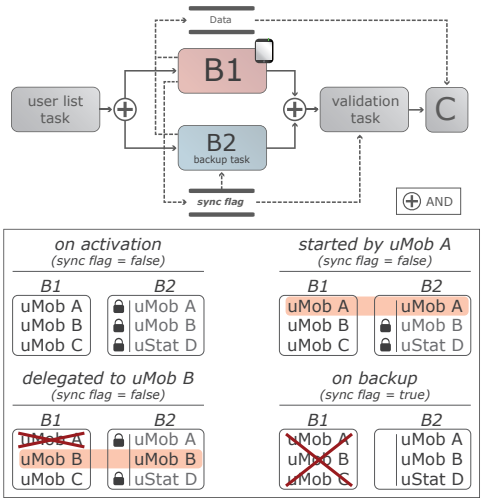


Figure 7. Complex backup operation

to the simple backup operation, for which the user list of B2 is determined after completing B1, this procedure speeds up user assignment.
Finally, Fig. 8 presents the scenario in which a binding of duties exists between two *mobile tasks* M1 and M2. Furthermore, both tasks write data and are not flagged as urgent. As illustrated by Fig. 8, adding the authorization constraint will have no implication for using the backup operation. The same applies to the other entailment constraints.

E. Delegation Service

During task activation and task delegation, all actions required to robustly execute a *mobile task* are performed automatically, coordinated by the *Mobile Delegation Service* (MDS). Since this delegation service maintains several lists with respect to user management, we first describe them before presenting the MDS.

User List Management: To foster robust execution of *mobile tasks*, our approach maintains three user lists: ul_{init}, ul_{mob}, and dl_{mob}. User list ul_{init} contains all mobile users authorized to perform a *mobile task* `t`. Based on ul_{init}, the mobile user list ul_{mob} is determined. Thereby, only those mobile users are added to ul_{mob}, which are currently online, whose user location is equal to the location of `t`, and who are not excluded by the filter defined during instantiation time. Then, ul_{mob} is used for assigning *mobile tasks* to mobile users. Further, if a *mobile task* `t` must be delegated, a mobile delegation list dl_{mob} will be determined. Thereby, all users from ul_{mob} are added to this list. Then, all users from list

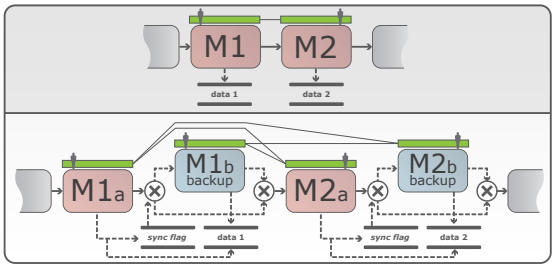


Figure 8. Mobile tasks constrained by a binding of duties and writing data

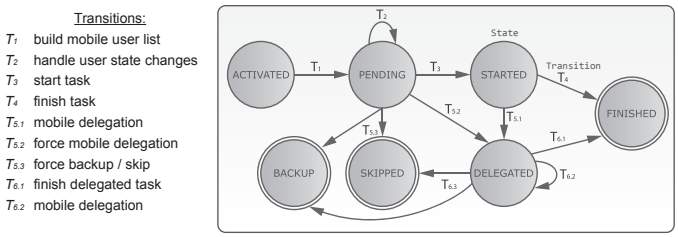


Figure 9. Mobile delegation service flow during runtime

dl_{mob} will be ordered from high to low priority. A low priority for a mobile user is assigned to him if his *battery status* is low or his *instant shutdown counter* is high. Both ul_{mob} and dl_{mob} will be recalculated if the connectivity status of a user from list ul_{init} changes.
Taking these lists into account, the mobile delegation service may enter six different states, denoted as `t(<STATE>)`, via respective state transitions T_i (cf. Fig. 9). Note that the delegation service starts when a *mobile task* `t` becomes activated.

The following scenarios are relevant, when taking urgency (timeout) to_u ($to_u = 0$ denotes a timeout), user list threshold th_{mul}, and the ability to skip a *mobile task* `t` into account:

1) **Normal task execution:** $user_a \in ul_{mob}$ starts *mobile task* `t` and performs it.
`t(PENDING)` $\rightarrow$ T_3 $\rightarrow$ `t(STARTED)` $\rightarrow$ T_4 $\rightarrow$ `t(FINISHED)`
2) **Delegated task execution:** $user_a \in ul_{mob}$ starts *mobile task* `t`. When the state of $user_a$ changes to *offline* or $to_u = 0$ holds, `t` will be automatically delegated to a user $user_b \in ul_{mob}$. The latter will then finish this task.
$T_3 \rightarrow$ `t(STARTED)` $\rightarrow T_{5.1} \rightarrow$ `t(DELEGATED)` $\rightarrow T_{6.1} \rightarrow$ `t(FINISHED)`
3) **Forced delegation:** Forced delegation becomes necessary, if `t(PENDING)` $\wedge$ ($|ul_{mob}| < th_{mul} \vee to_u = 0$), or if `t(DELEGATED)` $\wedge$ ($to_u = 0 \vee$ State($user_b$) changes to *offline*), `t` must be delegated to another $user_n \in ul_{mob}$
$T_{5.2} \rightarrow$ `t(DELEGATED)` $\vee\ T_{6.2} \rightarrow$ `t(DELEGATED)`
4) **Skip or Backup:** Skip or backup will be performed, if `t` changes into one of these scenarios: if (`t(PENDING)` $\wedge$ $to_u = 0 \wedge |ul_{mob}| = 0$) $\vee$ (`t(DELEGATED)` $\wedge$ $to_u = 0$ $\wedge$ $|dl_{mob}| = 0$). Furthermore, if `IS_SKIPPABLE(t)` = true, `t` will transits to `SKIP`, otherwise to `BACKUP`
(`t(PENDING)` $\rightarrow T_{5.3} \rightarrow$ `t(SKIP)` $\vee$ `t(BACKUP)` / `t(DELEGATED)` $\rightarrow T_{6.3} \rightarrow$ `t(SKIP)` $\vee$ `t(BACKUP)`)

IV. Entailment Constraints For Mobile Tasks

This section discusses how our approach realizes the entailment constraints and how it combines them with its mobile delegation service. Basically, the provided backup operations as well as the ability to skip tasks ease the integration of entailment constraints and their handling during process execution. In particular, we can ensure that all mobile tasks will be properly executed, satisfying the defined constraints. In order to integrate entailment constraints into our approach, we enhance our mobile delegation service. In this context, consider the scenario shown in Fig. 10. It shows two mobile tasks M_1 and M_2 constrained by a separation of duties. Further, consider the delegation lists for both mobile tasks and assume that during runtime delegations to all users authorized to perform

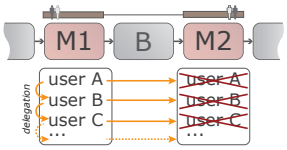


Figure 10. Mobile delegation service combined with separation of duties

M_1 become neccessary. As a result, no mobile user may perform M_2 later on since the delegations applied to M_1 will remove all mobile users from the user list of M_2. Then, the backup service ensures that M_2 will be performed.
In order to address such scenarios properly, we apply several changes to the user list (dl_{mob}) management of our mobile delegation service. We refer to Fig. 13, which shows the overall approach for these changes.

Separation of Duties: In Fig. 13, ① shows how we change the order of the user list of M1 in order to realize separation of duties more properly. First, we compare the user lists of the constrained tasks M1 and M2 (cf. Fig. 13; *compare(M1,M2)*). Then, for the user list of M1, a low priority is assigned to the users also appearing in the user list of M2 (cf. Fig. 13; *setLowPriority(M1.A,M1.B)*). Fig. 13, ① depicts the scenarios before and after the change.

- **Before change:** If delegations to the first two users A and B become necessary for M1 during runtime, only user D will later be allowed to perform M2 due to the separation of duties constraint between M1 and M2.
- **After change:** By contrast, if delegations to the first two users C and A become necessary for M1 during runtime, there still exist two users B and D who may perform M2. Compared to the situation *before change*, we obtain one additional mobile user for delegation purpose.

Binding of Duties: Fig. 13, ① shows the change we make in order to realize *binding of duties* appropriately. Opposed to *separation of duties*, we do not change user lists. Instead, we cope with the situation that no mobile user satisfies the binding of duties constraint. In order to illustrate this, consider the situations *before* and *after change*. Before change, the user lists for M3 and M5 indicate that *binding of duties* cannot be enforced since no mobile user is allowed to perform both tasks. As a result, in this scenario, M3 as well as M5 will be executed using the backup service. To be able to still execute M3 and M5 on a mobile device, we apply the following changes: First, we allow every mobile user to perform M3 (cf. Fig. 13; after change; *select(M3)*). Then, our backup service is applied to M5 (cf. Fig. 13; after change; *backup(M5)*). In this context, we use the complex backup operation to ensure that M5 will be executed properly and quicker than with the simple backup operation. We realize two ways for performing the backup task: *First*, we assign it to the user who has performed M3. In addition, she must perform it on a stationary computer. *Second*, we allow all mobile users, authorized to perform M5, to execute the backup task. Following this, the validation task of the backup service is performed, and the user who has performed M3, must confirm that the task was performed correctly.

Cardinality: Regarding the cardinality constraint (cf. Fig. 13 ④), no changes are required.

After integrating entailment constraints with our approach, the question emerges whether the combined use of several constraints in the context of a business process necessitates additional considerations.

Binding of Duties combined with Separation of Duties: In order to combine *binding of duties* with *separation of duties* (cf. Fig. 13 ②), no extensions are required. For example, consider the scenario depicted in Fig. 13, ②. Since the *separation of duties* constraint between tasks M2 and M5 has no effect on the *binding of duties* constraint between M1 and M2, no extensions are required.

Separation of Duties combined with Binding of Duties: Fig. 13, ③ shows how we adapt the order of the user list of M1 to properly support the combined use of *separation of duties* with *binding of duties*. First, we compare user lists of mobile tasks M1, M2, and M5 (cf. Fig. 13; *compare(M1,M2,M5)*). Second, we adapt the user list of M1 (cf. Fig. 13; *setLowPriority(M1.A)*). Fig. 13, ③ depicts the scenarios before and after change.

- **Before change:** If a delegation to user A becomes necessary for M1 during runtime, the *binding of duties* for M2 and M5 cannot be enforced.
- **After change:** After adapting the user list of M1, delegations to users B and C are still possible for M1 during runtime without affecting the *binding of duties*.

Cardinality combined with Binding of Duties: Fig. 13, ⑤a, shows how *cardinality* and *binding of duties* constraints can be combined. In this context, we assign a *binding of duties constraint* to all task instances of M1 (cf. Fig. 13; *add.BindingOfDuties*) and M2; i.e., each instance of M1 and M2 will be performed by the same mobile user.

Cardinality combined with Separation of Duties: Fig. 13, ⑤b, shows how *cardinality* and *separation of duties* constraints are combined. We provide two options for this: *First*, each task instance of M1 may be constrained through *separation of duties* with M2 and then threshold th_{mul} be changed for all task instances (cf. Fig. 13; *adjuste.Threshold*). The latter ensures that not all users, who may work on an instance of M1, will be a subject for delegation. As a result, the number of mobile user who may perform M2 increases. *Second*, we define a *binding of duties* constraint between all task instances of M1 and combine it with a *separation of duties* constraint with M2 (cf. Fig. 13; *add.BindingOfDuties*); i.e., all task instances of M1 are performed by the same user, who is then not allowed to perform M2.

We discuss two cases of particular interest. The first one is illustrated by Fig. 11. Consider task M2. It is located between M1 and M3, which are constrained by a *binding of duties*. In this context, the question emerges whether changes made to M2 will affect the separation of duties constraint between M1 and M3. Assume that user A performed M1. Then, he must perform M3 as well. However, if he is also authorized to perform M2, the probability that he will be able to perform M3 might decrease. Since his mobile device consumes power while he is working on M2, he might be unable to directly perform M3 afterwards due to a low battery status. Therefore, we decrease the priority of this user in the respective user list of M2.

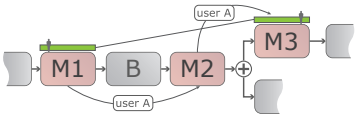


Figure 11. Mobile tasks M1 and M3 constrained by a binding of duties and preceding/succedding mobile task M2

Another special case is shown in Fig. 12. Between mobile tasks M1a and M2a a *binding of duties* constraint exists. Furthermore, both tasks produce data. Recall that for a task producing data, the presented backup service ensures that it will be executed properly. Furthermore, assume that for M1a the backup service must perform M1b during runtime. Then, it must be determined who shall be allowed to perform M2a. If the *binding of duties* constraint between M1a and M2a is strictly enforced, M2a must be automatically performed applying the backup service to M2b, since no mobile user has performed M1b. In this particular situation, we also consider the location attribute of M2a and change the execution semantics of the backup service. To assess the location, we need to meet the demands of mobility properly. In this context, the changes we make with respect to the scenario from Fig. 12 are as follows. *First*, if a location is defined for M2a, we allow arbitrary mobile users to perform M2b. Furthermore, the validation task of the backup operation for M2b is used to confirm that M2b was performed correctly. In turn, this confirmation must be done by the user performing M1b. However, applying this change means that we violate the *binding of duties* constraint between M1b and M2b on one hand and allow mobile users to perform a backup task on the other. With the latter, we enhance overall mobility. Based on the validation task, the *binding of duties* is satisfiable through user confirmation. *Second*, if no location is defined for M2a, we apply the same procedure as described above. In addition, M2b may then be also performed by stationary users since no particular location is defined. Overall, these changes allow for a better handling of the scenario from Fig. 12 and further fosters mobility.

V. Validation

User acceptance is often neglected in the context of mobile process and task support. To demonstrate the general feasibility of our approach on one hand and to assess user acceptance on the other, we implemented a proof-of-concept. In this context, we applied the procedure depicted in Fig. 5. As a result, the core of our prototype is implemented as intermediate component between a process engine and mobile devices. Moreover, it is based on existing tools and standards.

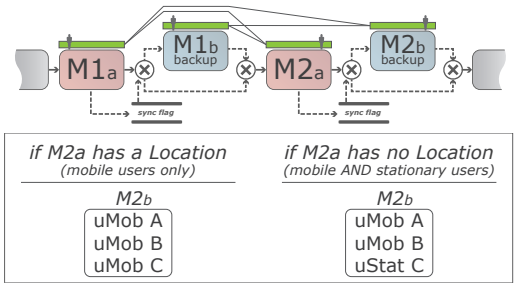


Figure 12. Binding of duties and backup operations

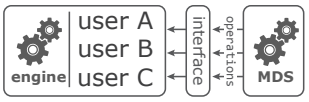


Figure 14. Integrating entailment constraints with prototype

Implementing a specific process engine, which provides all functions to create and execute mobile tasks, have constitute a possible direction as well. However, if a process management system is already in use, the introduction of another engine might be not accepted (e.g., due to high efforts for transfering process models to the new engine). Therefore, our approach provides an engine-independent interface for executing mobile tasks. Since invocation of web services is a core feature of any modern process engine, we have realized a service-driven approach for this purpose.

Due to lack of space, we only sketch how we integrated entailment constraints into this prototype (cf. Fig. 14): Based on the service-driven interface provided, the mobile delegation service gets access to an existing user management. Recall that we integrate process engines already in use including their user management. Through the access to such an existing user management, the mobile delegation service is run in two stages. First, it computes the mobile user and delegation lists. Second, it applies changes (cf. Fig. 14) to mobile user and delegation lists as described in Section IV. Due to lack of space, we omit details on the architecture of the prototype.

VI. Related Work

Two research fields are relevant in the context of our work: approaches addressing mobile process execution in general and approaches integrating entailment constraints with business process execution.

In the first category, work exists providing logical models for mobile processes on one hand and architectures and implementations of light-weight process engines on the other. Logical models for mobile processes include, for example, approaches partitioning BPEL process models and executing the resulting process fragments on mobile devices [12]–[15]. Similar approaches exist in the context of distributed process execution (e.g., [16]). However, none of them provides support for executing mobile tasks as in our approach.

There exist specific approaches handling failures like broken connections or missing communication facilities [14], [17]–[22]. Corresponding implementations mainly apply web service standards and rely on BPEL (or more specific execution models). In turn, none of them provides self-healing techniques to relieve users from manual tasks in case of errors. Note that this is crucial with respect to overall user acceptance. Generally, self-healing techniques, like the backup service we suggest, and task migration support, like the presented delegation service are indispensable when executing mobile tasks in the large scale while targeting at a high user acceptance at the same time. Existing approaches addressing respective aspects in the context of process-aware information systems again focus on BPEL as execution language [13], [23], [24].

Several approaches exist integrating entailment constraints with business process execution [9], [25]–[27]. They all focus on the specification of entailment constraints and their satisfiability. Thereby, those approaches dealing with delegation in the context of entailment constraints are directly related to

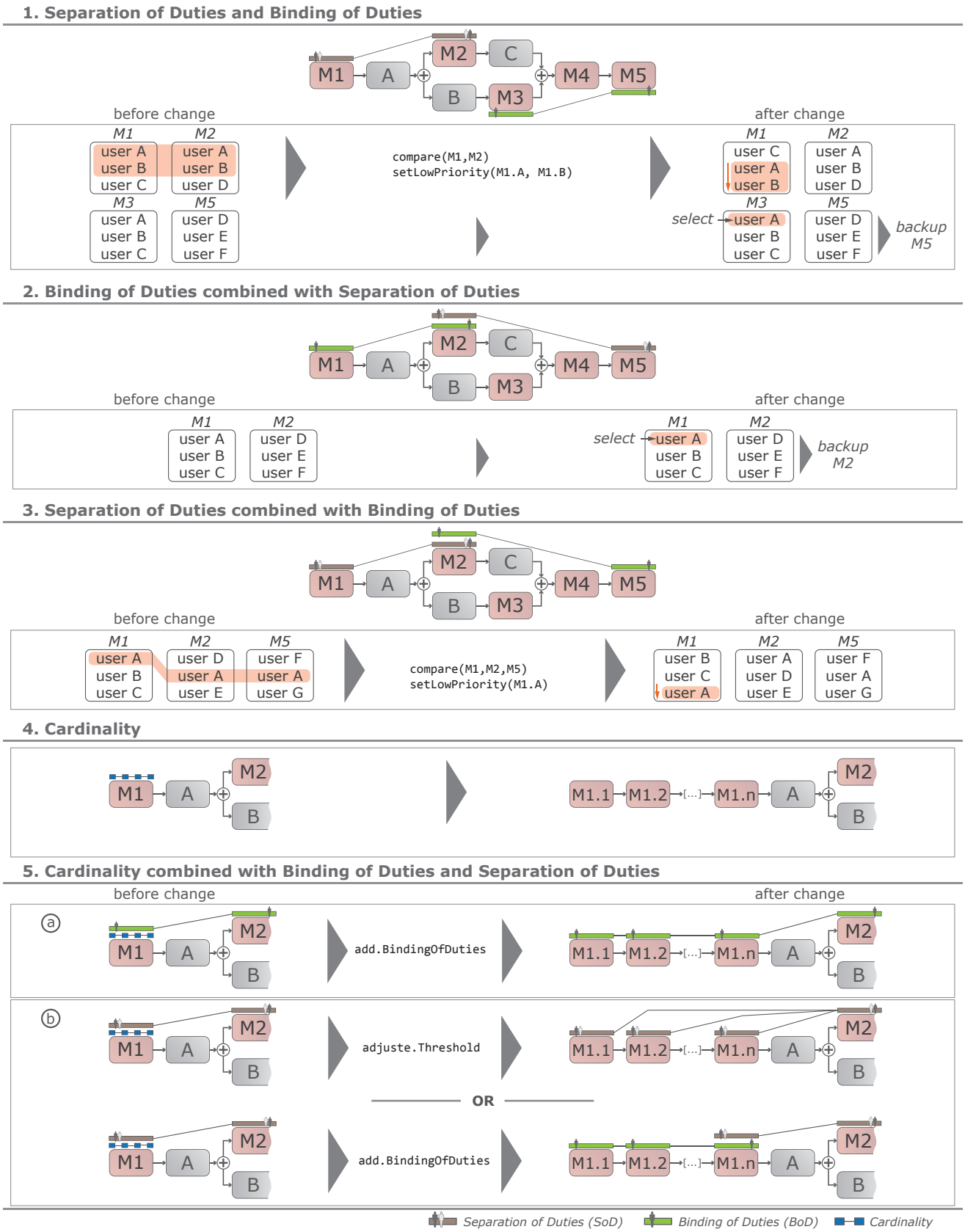


Figure 13. Enabling entailment constraints for mobile process execution

our approach [25], [28]–[31]. In turn, they use delegation in a different way, delegating tasks in a controlled manner from one user to another based on defined context configurations. Usually, a delegator requests a delegation to a delegatee. Usually this is accomplished in a user-centric way. However, to use delegation as a strategy for delegating tasks automatically in case of errors has not been addressed yet. Finally, to the best of our knowledge, there is no work dealing with the integration of entailment constraints and mobile process execution.
Altogether, the integration of entailment constraints with mobile process and task support has not yet received the attention as in our approach.

VII. Summary

We introduced an approach for integrating entailment constraints with mobile process and task support. In this context, the presented backup service as well as the mobile delegation service allow for a robust process execution. Our overall vision is to provide advanced mobile process and task support. The integration of entailment constraints constitutes one important step towards enhancing our approach. Mobile users are often in a line of visibility and hence separating or binding responsibilities might be crucial. To meet this demand, we added *separation* and *binding of duties* constraints to our approach. Further, in a mobile context, tasks may have to be executed more than once. This is covered by the cardinality constraint in our approach. Moreover, we argued that integrating these constraints must not harm overall process execution and we showed how this can be ensured based on the presented backup service. We further described changes to our mobile delegation service to optimize the support of entailment constraints. In particular, this optimization aims at decreasing the use of the backup service on one hand and fostering mobility on the other. Overall, we have shown that integrating entailment constraints with mobile process and task support is feasible. In future work, we will make experiments in the field, e.g., considering scenarios in which entailment constraints exist between mobile and stationary tasks.

References

[1] J. Schobel and M. Schickler and R. Pryss and H. Nienhaus and M. Reichert, "Using Vital Sensors in Mobile Healthcare Business Applications: challenges, Examples, Lessons Learned," *Int'l Conference on Web Information Systems and Technologies*, pp. 509–518, 2013.

[2] M. Reichert and B. Weber, *Enabling Flexibility in Process-Aware Information Systems: Challenges, Methods, Technologies*. Springer, 2012.

[3] ——, "Process Change Patterns: Recent Research, Use Cases, Research Directions," *Seminal Contributions to Information Systems Engineering - 25 Years of CAiSE*, pp. 398–404, 2013.

[4] J. Kolb and M. Reichert, "A Flexible Approach for Abstracting and Personalizing Large Business Process Models," *Applied Computing Review*, vol. 13, no. 1, pp. 6–17, March 2013.

[5] A. Lanz and B. Weber and M. Reichert, "Time patterns for process-aware information systems," *Requirements Engineering*, pp. 1–29, 2012.

[6] B. Weber and M. Reichert and S. Rinderle-Ma, "Change Patterns and Change Support Features - Enhancing Flexibility in Process-Aware Information Systems," *Data Knowl. Eng.*, vol. 66, no. 3, pp. 438–466, 2008.

[7] R. Pryss and D. Langer and M. Reichert and A. Hallerbach, "Mobile Task Management for Medical Ward Rounds - The MEDo Approach," *Proc. BPM'12 Workshops, LNBIP*, no. 132, pp. 43–54, 2012.

[8] R. Lenz and M. Reichert, "IT Support for Healthcare Processes - Premises, Challenges, Perspectives." *Data Knowl. Eng.*, vol. 61, no. 1, pp. 39–58, 2007.

[9] J. Crampton, "An algebraic approach to the analysis of constrained workflow systems," *Proc. 3rd Workshop on Foundations of Computer Security*, pp. 61–74, 2004.

[10] R. Pryss and J. Tiedeken and U. Kreher and M. Reichert, "Towards Flexible Process Support on Mobile Devices," *Proc. CAiSE'10 Forum*, pp. 150–165, 2010.

[11] R. Pryss and J. Tiedeken and M. Reichert, "Managing Processes on Mobile Devices: The MARPLE Approach," *CAiSE'10 Demos*, June 2010.

[12] L. Baresi and A. Maurino and S. Modafferi, "Workflow partitioning in mobile information systems," *Proc. IFIP TC8 Working Conf. on Mobile Information Systems*, pp. 93–106, 2004.

[13] L. Baresi and S. Guinea and L. Pasquale, "Self-healing BPEL processes with DYNAMO and the JBoss rule engine," *Int. Workshop on Engineering of Software Services for Pervasive Environments*, pp. 11–20, 2007.

[14] K. Hahn and H. Schweppe, "Exploring Transactional Service Properties for Mobile Service Composition," *MMS'09*, pp. 39–52, 2009.

[15] H. Schmidt and F. J. Hauck, "SAMPROC: middleware for self-adaptive mobile processes in heterogeneous ubiquitous environments," *Proc. 4th Middleware Doctoral Symposium*, pp. 1–6, 2007.

[16] T. Bauer and M. Reichert and P. Dadam, "Intra-subnet Load Balancing in Distributed Workflow Management Systems," *Int'l Journal of Coop. Inf. Sys.*, vol. 12, no. 3, pp. 205–323, 2003.

[17] C. P. Kunze, "DEMAC: A Distributed Environment for Mobility-Aware Computing," *Proc. 3rd Int. Conf. on Perv. Comp.*, pp. 115–121, 5 2005.

[18] G. Hackmann and M. Haitjema and C. Gill, "SLIVER: A BPEL Workflow Process Execution Engine for Mobile Devices," *ICSOC'06*, pp. 503–508, 2006.

[19] H. Schmidt and R. Kapitza and F. J. Hauck, "Mobile-process-based ubiquitous computing platform: a blueprint," *Proc. 1st Workshop on Middleware-application interaction*, pp. 25–30, 2007.

[20] G. Stuermer and J. Mangler and E. Schikuta, "Building a Modular Service Oriented Workflow Engine," *IEEE Int. Conf. on Service-Oriented Computing and Applications*, pp. 1–4, 12 2009.

[21] D. Battista and M. Leoni and A. Gaetanis and M. Mecella and A. Pezzullo and A. Russo and C. Saponaro, "ROME4EU: A Web Service-Based Process-Aware System for Smart Devices," *Proc. ICSOC'08*, pp. 726–727, 2008.

[22] S. Zaplata and V. Dreiling and W. Lamersdorf, "Realizing Mobile Web Services for Dynamic Applications," *I3E'09*, pp. 240–254, 9 2009.

[23] L. Baresi and S. Guinea, "DYNAMO and Self-Healing BPEL Compositions," *Proc. 29th Int. Conf. on Software Eng.*, pp. 69–70, 2007.

[24] S. Zaplata and K. Kottke and M. Meiners and W. Lamersdorf, "Towards Runtime Migration of WS-BPEL Processes," *WESOA'09*, 4 2010.

[25] J. Crampton and H. Khambhammettu, "Delegation and satisfiability in workflow systems," *Proc. 13th ACM Symposium on Access Control Models and Technologies*, pp. 31–40, 2008.

[26] E. Bertino and E. Ferrari and V. Atluri, "The specification and enforcement of authorization constraints in workflow management systems," *ACM Trans. on Inf. and Sys. Sec.*, pp. 65–104, 1999.

[27] C. Wolter and A. Schaad and C. Meinel, "Task-based entailment constraints for basic workflow patterns," *Proc. of the 13th ACM symposium on Access control models and technologies*, pp. 51–60, 2008.

[28] L. Zhang and G-J. Ahn and B-T. Chu, "A rule-based framework for role-based delegation and revocation," *ACM Trans.on Inf. and Sys. Security*, vol. 6, no. 3, pp. 404–441, 2003.

[29] K. Gaaloul and F. Charoy, "Task delegation based access control models for workflow systems," *Software Services for e-Business and e-Society*, pp. 400–414, 2009.

[30] V. Atluri and J. Warner, "Supporting conditional delegation in secure workflow management systems," *Proc. 10th ACM Symposium on Access Control Models and Technologies*, pp. 49–58, 2005.

[31] L. Zhang and G-J. Ahn and B-T. Chu, "A rule-based framework for role-based delegation and revocation," *ACM Transactions on Information and System Security (TISSEC)*, vol. 6, no. 3, pp. 404–441, 2003.