Migratability of BPMN 2.0 Process Instances*

Angineh Barkhordarian, Frederik Demuth, Kristof Hamann, Minh Hoang, Sonja Weichler, and Sonja Zaplata

Distributed Systems and Information Systems, Department of Informatics
University of Hamburg, Germany
hamann@informatik.uni-hamburg.de

Abstract. The migration of running process instances allows for a dynamic distribution of individual business processes at runtime. However, a widely-used standardized process description language and an agreed format for the exchange of process instance data are vital for the applicability of such concept. The newly evolved standard of the *Business Process Model and Notation (BPMN 2.0)* is currently gaining acceptance in many organizations and is supported by a growing number of process engines. In order to leverage BPMN for the dynamic distribution of business processes, this paper presents an analysis on the migratability of running BPMN process instances. The results include a mapping of BPMN 2.0 control flow elements to an existing migration model and a novel migration concept for process instances which contain BPMN-specific elements such as events, pools and user tasks. In addition, the effort for extending a BPMN process engine is evaluated by a prototype implementation based on the open source *Activiti process engine*.

1 Motivation

In today's dynamic environments, business processes are often subject to changes related to the content and the structure of the business case. In consequence, flexibility of supporting information and communication systems is one of the most driving factors. Considering that a business process is not always executed by a single organization or, more technically, by a single process engine, also the requirements for the distributed execution of individual process instances can change dynamically. A typical example is the spontaneous shift of a selected process partition to a mobile device in order to allow for its offline execution in a different location [3,13]. Other examples include the dynamic distribution of process instances due to load balancing strategies for process engines [12] or the runtime exchange of business partners in order to quickly react to market changes or to individual demands of customers [11].

In situations where requirements for a distributed execution of individual process instances can change dynamically, a flexible distribution mechanism is needed. In comparison to other distribution models which require to determine

* The research leading to these results has received funding from the European Community's Seventh Framework Programme under grant agreement 215483 (S-Cube).

process partitions at design time (e.g. web service choreographies), the concept of *process instance migration* allows for deciding about most distribution parameters at the runtime of each process instance [6,10,14]. The procedure used here basically involves stopping the execution of a running process instance, capturing its current status, and transferring both the process model and its instance data to another process engine where the execution of the process is continued. Using this strategy, a business process can be distributed at nearly any time with an arbitrary granularity of process partitions and open number of potentially participating process engines [14].

As such dynamic distribution requires a common understanding of the business case as well as a standardized specification including an execution semantic, the applicability of process instance migration is dependent on the underlying process description language, its specific structure and its control flow elements. A promising candidate is the newly evolved standard of the *Business Process Model and Notation (BPMN 2.0)* [9] which is – in contrast to its predecessor BPMN 1.1 – not limited to a graphical representation of business process models but also provides an execution semantic. It therefore closes the gap between business process analysts and technical developers and thus significantly reduces the time between the development phases of design and implementation. Although BPMN also integrates elements to describe distribution (i.e. *pools* and *lanes*), a runtime migration of BPMN process instances has not yet been considered. Instead, the distribution of BPMN processes is still an inflexible and time-consuming procedure because it requires the manual deployment of predetermined process partitions on pre-selected process engines prior to runtime.

In order to address these issues, this paper presents an analysis on the general *migratability* (i.e. the ability to partition and transfer the process within a specific control flow structure) of running BPMN process instances based on an existing generic migration model which allows the representation of language-independent process instance data. Relevant background information about the migration model and related approaches are presented in Section 2. Section 3 discusses the specific characteristics of BPMN language elements and proposes a mapping of these elements to the migration model. Section 4 evaluates the effort for enhancing an existing BPMN process engine in order to realize the migration within a prototype implementation. Results are summarized in Section 5.

2 Background and Related Work

Process instance migration as a basic concept for distributed workflow management (not to be confused with migration of instances to another schema, e.g. *case migration* [5]) has been introduced by Cichocki and Rusinkiewicz [6] in 1997. More recently, the framework *OSIRIS* [10] relies on passing control flow between distributed workflow engines in order to execute service compositions. In *Adept Distribution* [4] a similar approach to process fragmentation is presented which supports dynamic assignment of process parts to so-called *execution servers*. The control of a particular process instance migrates from one execution server

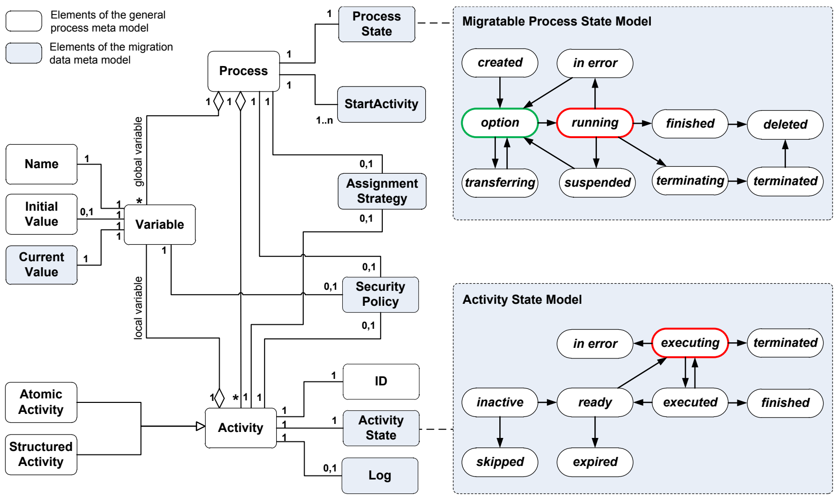

Fig. 1. Generic migration model (simplified) [14]

to another and process activities can influence the next participant. Related to this, Atluri et al. [1] present a process partitioning algorithm which creates self-describing subprocesses allowing dynamic routing and assignment. Process migration has also particularly been applied to the area of mobile process execution, e.g. by Montagut and Molva [8]. Their approach relies on passing control flow between distributed WS-BPEL engines in order to access applications internal to mobile devices. Similarly, the *DEMAC* middleware [13] is able to delegate process execution (in whole or in part) to other stationary or mobile process engines. However, the presented approaches either propose a *migration-specific extension* of a (standard) process description language such as XPDL or WS-BPEL (e.g. [8,13]) or use a *completely proprietary specification* (e.g. [6,10,4,1]). Based on its novelty as an executable process description language, migration of (unmodified) BPMN process instances has not been considered yet.

As a more general approach, a technology-independent migration model has been introduced in [14] (cp. Figure 1). Here, the migration model only assumes the existence of a common minimal process model consisting of a finite number of *activities* representing the tasks to be fulfilled during process execution, and a finite number of global or local *variables* holding the data used by these activities. Activities are either *atomic activities* (i.e. represent a specific task) or *structured activities* (i.e. a control flow structure as a container for other activities). A process description complying to these properties can be supplemented with a separate description of migration data documenting the state of the variables (*current value*) as well as the execution state of the process (*process state*) and of each activity (*activity state*). In order to preserve the process's consistency and the integrity of its data, a process instance is not allowed to be migrated until all of the currently executed atomic activities are completed. Thus, the process lifecycle state *option* defines a stable point to transfer a process during its execution which can only be reached if none of the activities are in the activity lifecycle state *executing*. In order to control the selection of target execution systems, each activity of the process can be optionally connected to an *assignment strategy* and/or a *security policy* representing user defined distribution strategies and restrictions. In addition, a set of activities can be referenced as *start activities* to mark the first activities to be executed after process migration. In order to connect the migration data model with the original process model, each activity and variable must have a unique name or identifier (ID). As BPMN complies to these basic requirements, this generic migration model can be used as a basis to analyse the migratability of BPMN process instances.

The concept of process instance migration has recently been evaluated for basic elements of XPDL and WS-BPEL processes (cp. [14]). However, regarding the ability for migration, BPMN processes have inherently different characteristics and thus impose new challenges for migration. In contrast to WS-BPEL, BPMN processes are able to combine interactive user-centric tasks and automatic application tasks such as web services on both a specific or an abstract level. Furthermore, BPMN processes can already specify descriptions about a distributed execution and BPMN includes a strong event concept allowing event-based control flow constructs. In order to address these differences in more detail, the following section discusses relevant BPMN control flow elements with respect to a potential migration of the process.

3 Migrating BPMN Processes

The *Business Process Model and Notation* (*BPMN*) is a graph-based description language for the specification of business process models which can be expressed in a standardized graphical notation, and, since BPMN 2.0 [9], also in a respective executable XML representation. Its goal is to provide a single specification for notation, metamodel and interchange format of business processes. Following BPMN, a process consists of a number of flow objects (*tasks*, *gateways* and *events*), connecting objects (*sequence flows* and *message flows*), distribution objects (*pools* and *swimlanes*) and other artifacts (e.g. *data objects* and *groups*) (cp. Figure 2 for an overview). The migratability of the most important control flow constructs is discussed in the following.

3.1 Tasks

A BPMN process consists of at least one functional *task* which represents a single unit of work to be done. Considering the migration model presented in Section 2, a *task* corresponds to an *atomic activity*. However, in contrast to e.g. WS-BPEL activities, BPMN tasks can be defined on different levels of abstraction and do not require to bind a resource (e.g. a web service or a human process participant) by a static binding. Instead, resources can also be specified by a role or a number of characteristics and thus can be rebound after migration. Depending on the assignment strategy, a task can be rebound to a locally-available resource at the target system, the local binding can be replaced by a unique systemwide reference

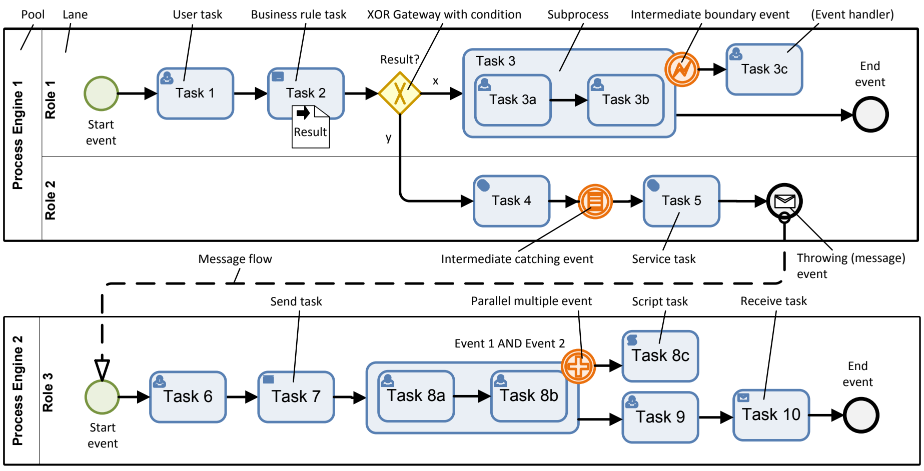

Fig. 2. Example of BPMN elements and control flow constructs

or, if applicable, the resource can be moved or copied to the target system [7]. In consequence, this abstraction implies more possibilities for migration and thus possibly leads to a higher level of flexibility.

In more detail, BPMN supports different types of tasks. *Service tasks* (calling a software application) and *manual tasks* (signaling a user to fulfill a task outside the computer system) can be executed by a remote process management system if the required resources are (locally or remotely) available. However, rebinding resources requires a common understanding such as a shared organizational model or an ontology which is not part of BPMN. In consequence, the migration of BPMN processes relying on such (abstract) constructs requires an additional agreement on specific technologies to be used.

Furthermore, user interfaces for presenting process data and accepting the user's input are often process-specific and are deployed together with the process (e.g. an HTML form). The same is true for small code snippets which are carried along with the process. As such resources can be (physically) moved to the target system, migration of respective *user tasks* and *script tasks* is possible if the target system provides a compatible platform. However, the migration of arbitrary executable code (as with the *script task*) implies potential security risks and should thus be restricted to trustworthy partners only (cp. [14]).

Another restriction is required for messaging tasks: If *send task* and *receive task* belong to a pair of asynchronous communication (e.g. tasks 7 and 10 in Figure 2), such corresponding tasks must be executed on the same process engine in order to allow the answer message to be correctly delivered. This can be realized by applying a dynamic assignment strategy.

Finally, the specification of business rules is also outside the scope of BPMN. If business rule management systems are compatible and a *business rule task* is rebound to a local rule engine, it must be considered that the outcome may be different and thus may lead to a different control flow (cp. task 2 in Figure 2).

Table 1. Properties of throwing event types in BPMN 2.0

Type of event	Purpose of the event type	Latest point of subscription
top level start	instantiate process models	process deployment
event sub-process start	launch indepent sub-processes	start of surrounding (sub-)process
intermediate boundary	handle events in bounded flow objects	start of bounded flow object
intermediate catching	hold control flow until event occurs	control flow arrives at event

3.2 Gateways

A *gateway* is a control flow construct which determines forking and merging of alternative or parallel process paths depending on a set of (optional) conditions. With respect to the migration model, it is advantageous to consider a *gateway* to be a kind of atomic activity executed by the process engine itself. Now, the act of evaluating a condition can be assigned to a specific process engine (or a respective role) and thus it can be influenced which process engine should be responsible for its execution. Following this idea, a gateway is in the state *ready* if the process's control flow reaches the gateway, it is in the state *executing* while evaluating all of its conditions and is *executed* resp. *finished* if the outcome has been computed. Depending on the result, the states of the upcoming activities are set to *ready* or *skipped* (used by *dead path elimination*, cp. [14]). In consequence, migration is also possible during the execution of alternative or parallel paths.

3.3 Subprocesses

A *subprocess* represents a block activity containing an enclosed control flow which is executed in place of the subprocess. Therefore, it can be mapped to a complex activity in the migration model which enables migration at any time before, during or after its execution w.r.t. the state and migratability of enclosed elements. However, the description of the subprocess is required to be available at the target system. This is given if the calling process and the subprocess are specified within the same process description which is supported by both the graphical and the XML representation of BPMN (cp. task 3 in Figure 2).

3.4 Events

BPMN distinguishes events which are *thrown* by the process instance and events which are *caught* by the responsible process engine. Regarding migration, throwing an event is uncritical because it corresponds to a simple (atomic) activity which is performed by the process engine. In contrast, catching events involves the subscription for the respective events which are specified within the control flow, receiving the actual event instances, and starting the event handler (set of flow objects). In general, a (non-recurring) event can only be unsubscribed if the event has already been caught or the event-based control flow structure is finished. Considering the starting time and duration of a subscription for different event types (cp. Table 1), avoiding to split the procedure of *subscription*,

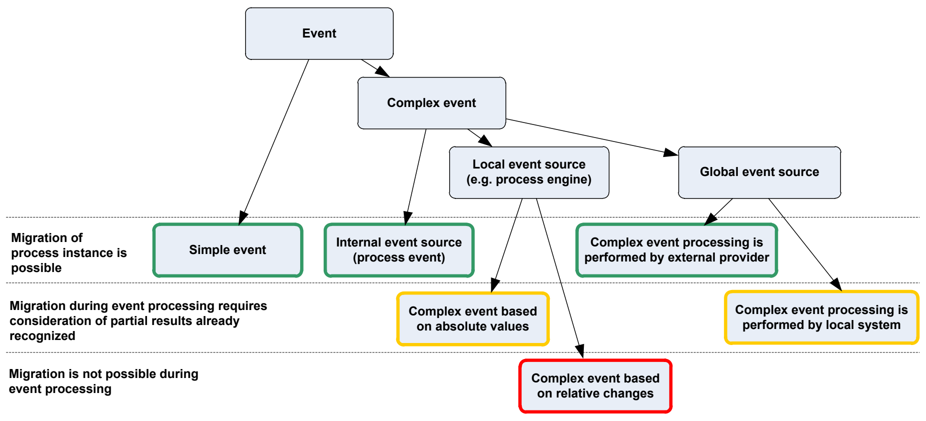
Event
Complex event
Local event source (e.g. process engine)
Global event source
Migration of process instance is possible
Simple event
Internal event source (process event)
Complex event processing is performed by external provider
Migration during event processing requires consideration of partial results already recognized
Complex event based on absolute values
Complex event processing is performed by local system
Migration is not possible during event processing
Complex event based on relative changes

Fig. 3. Classification of events with respect to a process instance migration

receiving and *processing* of events would therefore prohibit migration for all event-based control flow structures. Thus, a more flexible solution is required.

Figure 3 shows an overview of respective migration possibilities resulting from general considerations on event processing. Accordingly, Table 2 evaluates the specific events defined in BPMN 2.0. Non-complex events (e.g. *error*, *compensation*, *message*) are non-critical, because such simple events are processed atomically and, in case of migration, they are handled either on the source or target engine. However, a rebinding of the event source may be required. Complex events which are composed of process internal events (e.g. *(parallel) multiple events* using error and message events) are also non-critical, since they can be regarded as structured activities (cp. subprocesses) and the occurrence of a sub-event can be tracked by means of the activity lifecycle state. If the process instance requires subscription of events with an *external source*, it is necessary to take a closer look at the type of event and its relationship to the local environment. Complex events processed by an external provider are recognized by the process engine as a simple event and therefore inherit the corresponding migratability. If complex event processing is performed by the local system, sub-events which are already detected on the current process engine would have to be transferred to the target engine. Other complex events using local event sources from the context of the process engine (e.g. an event concerning the temperature in the local environment) are feasible, if the complex event is based on absolute values and sub-events are transferred on migration. However, in case of relative changes, it is possible that events are triggered by migration. An example is a complex event which is thrown if the temperature rises up to 150% of its average value. If the ambient temperature of the source and target location of process execution essentially differ during migration, the event may be thrown although the temperature has not changed at both locations at all. Throwing the event would thus be semantically incorrect and has to be avoided.

Table 2. Evaluation of specific events in BPMN 2.0

Event name	Purpose	Compl.	Int.	Loc.	Rem.	Migration
Compensation	compensation of activities	–	✓	–	–	possible
Error	hierarchical processing of errors	–	✓	–	–	possible
Escalation	hierarchical processing of escalations	–	✓	–	–	possible
Terminate	terminates the current process instance	–	✓	–	–	possible
Link	connects control flow inside a process	–	✓	–	–	possible
Signal	signaling inside and across processes	–	✓	✓	–	possible[1]
Cancel	canceling transactions	–	✓	✓	–	possible[2]
Message	receiving a message	–	–	✓	–	possible[1,3]
Timer	wait for a specific time period/point	–	–	✓	–	possible[4]
Conditional	reference to arbitrary conditions	*	*	*	*	depends on source/complexity
Parallel/Multiple	requires all/one of multiple events	*	*	*	*	depends on used event types

[1] source rebinding possibly required
[2] see message event in case of external transactions
[3] the message event is locally triggered on receiving a (remote) message
[4] time periods can be replaced by an absolute point of time on activation

A reliable solution is to consider receiving events as (atomic resp. structured) activities which require the process engine to perform event processing. The activity is set to the *ready* state if the process's control flow reaches the event resp. the first activity of its bounding (resp. surrounding) scope. At this point of time, also the event source has to be subscribed at the latest. The event activity goes to the state *executing* at the time event processing is started, i.e. a (partial) event is received. Simple events have the character of an atomic activity and are *executed* and *finished* right after the event has been detected and the upcoming activity has been activated. Complex events consist of a rule for pattern recognition of multiple event parts along a time period and remain in the state *executing* until the event is completely detected and processing is ended. In that case, the event is set to *executed* resp. *finished* and upcoming activities are activated. In doing so, complex event processing cannot be interrupted by a migration of a process instance as this would lead to a loss of partial results of event detection and, possibly, incorrect results. In case of boundary events, there is also the option to set the event to the states *expired* resp. *terminated* if the execution of the scope is finished before the event has (completely) occurred. The process engine can now unsubscribe from the event source.

3.5 Pools and Swimlanes

BPMN 2.0 allows to express the collaboration of multiple resources and organizational units. The *swimlane* element represents the assignment of flow objects to roles or specific participants and thus represents a *resource level view* of a distributed business process. The *pool* element is a higher level construct to express which business unit or technical unit is responsible for the execution of a specific process partition and for calling the required resources. Pool elements can thus be mapped to the migration model as a pre-defined *assignment strategy* for migration targets based on design-time distribution requirements. In consequence, the distribution specified in the collaboration among pools (i.e. *message* *flows*) can be used to represent the transfer of the process instance from one process engine to another. Process instance migration thus provides an interesting alternative interpretation of the distribution specified in BPMN's executable collaboration models. Basic strategies to also support a parallel execution of process partitions among different process engines can be found in [14].

4 Extension of the Activiti Process Engine

In order to estimate the efforts for extending an existing BPMN process engine with additional support for process instance migration, the open source Activiti process engine [2] was selected. In general, the procedure of process instance migration can be separated into four distinct phases:

1. Stop the process engine's execution of a process instance in a consistent state.
2. Extract and save runtime information, then delete the process instance or mark it as *transferring.*
3. Send process definition and runtime information to the target system.
4. Resume the transferred process instance on the target system.

Following this procedure, Activiti has been enhanced by an internal event concept in order to trigger a blocking event each time migration becomes possible, i.e. after an activity's execution has been finished. The new *migration handler* component handles such events by stopping the process execution if there is currently a demand for migration (i.e. based on user-defined migration strategies; phase 1). Otherwise, the execution is continued regularly. In the second phase, process instance data is retrieved from the process engine and is transformed according to the generic migration model. Activiti uses a relational database to store runtime information of processes in a proprietary schema, which is used by the migration handler in order to create a system-independent XML file containing the migration data. In phase three, this file and the (original) BPMN process model are transferred to another migration-aware (Activiti) BPMN engine by using a well-defined Web Service interface. After successful transmission, the target engine subscribes for local and remote events which are in the state *ready* and, subsequently, the source engine can unsubscribe for these events and delete the interrupted process instance. Finally, in the fourth phase, the transferred process instance is resumed. In Activiti, this is done by creating a new process instance and by applying the current state from the XML migration data.

The experiences with the prototype show, that, dependent on the process engine's architecture, deep intrusions in the engine's code may be necessary in order to gather the current process state and influence the process execution. However, the engineering effort for extending an existing BPMN process engine has to be performed only once in order to migrate arbitrary process instances automatically. Compared to a physical fragmentation which requires modification and deployment of each individual process model, this engineering effort is feasible – especially if many process instances have to be distributed individually.

5 Conclusion

Although BPMN 2.0 is developed as a process interchange format, the migration of running BPMN process instances holds several challenges. First, a meta model for process instance data is not part of BPMN. Second, the process description can be defined in a rather abstract way which requires rebinding of activities to new execution environments and resources. Based on an existing approach, this paper has thus proposed a mapping of BPMN 2.0 elements to a generic migration model and has discussed benefits and restrictions of the runtime migration of BPMN process instances. It shows that migration of BPMN process instances is – in general – possible. In order to ensure the original semantic of execution, migration has to be limited when tasks (including complex event processing) are currently executed. Due to its higher abstraction level, BPMN also allows for more flexibility in rebinding resources and events to the target system of migration. However, many relevant implementation details such as bindings to software applications, references to organizational models or the description and integration of user interfaces are not included within the BPMN standard and use additional (platform-dependent) instance data – which is still a challenge for ad-hoc process instance migration and has to be addressed in future work.

References

1. Atluri, V., et al.: A Decentralized Execution Model for Inter-organizational Workflows. Distrib. Parallel Databases 22(1), 55–83 (2007)
2. Baeyens, T., et al.: Activiti BPM Platform (2011), `http://www.activiti.org/`
3. Baresi, L., Maurino, A., Modafferi, S.: Workflow Partitioning in Mobile Information Systems. In: MOBIS 2004, pp. 93–106 (2004)
4. Bauer, T., Dadam, P.: Efficient Distributed Workflow Management Based on Variable Server Assignments. In: Wangler, B., Bergman, L.D. (eds.) CAiSE 2000. LNCS, vol. 1789, pp. 94–109. Springer, Heidelberg (2000)
5. Casati, F., Shan, M.C.: Dynamic and adaptive composition of e-services. Inf. Syst. 26(3), 143–163 (2001)
6. Cichocki, A., Rusinkiewicz, M.: Migrating Workflows. In: Advances in Workflow Management Systems and Interoperability, pp. 311–326. NATO (1997)
7. Fuggetta, A., Picco, G.P., Vigna, G.: Understanding Code Mobility. IEEE Transactions on Software Engineering 24(5), 342–361 (1998)
8. Montagut, F., Molva, R.: Enabling Pervasive Execution of Workflows. In: Collaborative Computing: Networking, Applications and Worksharing. IEEE (2005)
9. OMG: Business Process Model and Notation (BPMN), Version 2.0. Tech. rep., Object Management Group (OMG) (2011)
10. Schuler, C., Weber, R., Schuldt, H., Schek, H.J.: Scalable Peer-to-Peer Process Management - The OSIRIS Approach. In: ICWS, pp. 26–34 (2004)
11. Weske, M.: Business Process Management: Concepts, Languages, Architectures. Springer, Heidelberg (2007)
12. Wutke, D., Martin, D., Leymann, F.: A Method for Partitioning BPEL Processes for Decentralized Execution. In: ZEUS 2009, pp. 109–114. CEUR-WS.org (2009)
13. Zaplata, S., Kunze, C.P., Lamersdorf, W.: Context-based Cooperation in Mobile Business Environments. Bus. and Inf. Syst. Eng. (BISE) 2009(4) (October 2009)
14. Zaplata, S., et al.: Flexible Execution of Distributed Business Processes based on Process Instance Migration. Journal of System Integration (JSI) 1(3), 3–16 (2010)