2009

TUPLESPACE-BASED INFRASTRUCTURE FOR DECENTRALIZED ENACTMENT OF BPEL PROCESSES

Daniel Wutke
Institute of Architecture of Application Systems, University of Stuttgart

Daniel Martin
Institute of Architecture of Application Systems, University of Stuttgart

Frank Leymann
Institute of Architecture of Application Systems, University of Stuttgart

TUPLESPACE-BASED INFRASTRUCTURE FOR DECENTRALIZED ENACTMENT OF BPEL PROCESSES

Daniel Wutke, Daniel Martin, Frank Leymann[1]

Abstract
Business processes in WS-BPEL are a manifestation of the two-level-programming paradigm where remote-accessible Web services are composed to potentially complex orchestrations. WS-BPEL processes are executed by Workflow Management Systems that navigate through the process' activities and interact with the orchestrated services. While Web service technology enables interactions with remote services, process navigation is typically done in a centralized manner. Especially in scenarios of complex interactions between multiple distributed process participants, this way of process enactment has several drawbacks. In this paper, we outline those drawbacks and propose an alternative approach to execution of BPEL processes in a distributed, decentralized manner.

1. Introduction

The Service Oriented Architecture (SOA) [1] is an approach to integrating enterprise applications in a flexible and loosely coupled manner. SOA is built on the notion of services, which are realizations of self-contained business functions and provide a service requester with an abstract view on business functions. An important aspect of SOA is service composition, in this context also referred to as *service orchestration*.

The Web Service Business Process Execution Language (WS-BPEL or BPEL for short) [4] is the de-facto standard for describing workflow-like orchestrations of Web services. Following the two-level-programming paradigm [5], compound services, i.e. business functions, are composed to new composite applications which then again can be exposed as services to facilitate further composition. BPEL offers a range of activities comprising (i) activities for defining process control flow (e.g. *sequence*, *flow*, *switch*, *while*), (ii) activities describing interaction with Web services (e.g. *invoke*, *receive*, *pick*, …), and (iii) activities that provide the necessary glue between activities (e.g. *assign*). In addition, BPEL defines the concept of variables for storing process-internal data such as intermediate processing results on a per-instance basis. To enable process execution, a BPEL process model, once created, is deployed to a *Workflow Management System (WfMS)*; in case of BPEL, the WfMS exposes the process to potential users after process deployment through a Web service interface. Triggered by incoming user requests that match the conditions defined in the process model, processes are instantiated and their execution is started which comprises *continuous*

[1] Institute of Architecture of Application Systems, University of Stuttgart, Germany

evaluation of process control flow and *execution of activity implementations*. Process execution in BPEL is control-flow driven; process control flow is defined through so-called *structured activities* which, similar to control constructs known from other programming languages, define the order of activity execution or the flow of the program. As of today, process control flow is typically evaluated by the *navigator* component of a centralized WfMS. The term *centralized* refers to the workflow engine conceptually being regarded as one single entity. Note that while it is possible to provide an implementation of a WfMS that can be distributed among a number of nodes in a clustered environment [6], the general assumption still is that the clustered nodes operate within the same administrative domain, for instance the same department of an enterprise. During evaluation of process control flow, the navigator component of the WfMS decides based on the current state of a process instance which BPEL activities are to be executed next and starts their execution. This navigation process continues until process instance execution is completed. While *structured activities* define the order in which activities are executed *basic activities* reflect pieces of application code that are to be evaluated during process execution; two classes of basic activities can be distinguished according to whether they involve interaction with WfMS-external applications or not. Activities such as *assign* or *wait* are provided by the WfMS itself, hence their execution does not require any interaction with WfMS-external entities. Interaction activities, which interact with the actual compound application logic (i.e. the "business functions") orchestrated by the process such as *invoke* and *receive*, trigger an interaction of the WfMS with WfMS-external applications. Since, in case of BPEL, these applications are Web services [2, 3] which can be invoked remotely using potentially various network transport protocols (enabled through the mechanism of Web service bindings), BPEL allows for *distributed application logic*. Hence one can say that the current way of executing BPEL processes relies on centralized (i.e. local) execution of process control flow and distributed application logic through the use of Web service technology.

While the aforementioned way of executing BPEL processes is appropriate for a wide range of scenarios, it has severe drawbacks in other scenarios. These are discussed in Section 2 where we motivate our work through an example scenario and a discussion of the benefits resulting from decentralized execution of process control flow. In Section 3, we introduce the proposed approach to decentralized execution of BPEL processes and outline the process model and the system architecture; this comprises a description of (i) the functional building blocks of the system and (ii) how they interact in order to facilitate decentralized process execution. In addition, the supported spectrum of distributed process execution is presented and discussed in detail. Section 4 provides an overview of related work in the area of distributed workflow management systems, process fragmentation and process outsourcing. Section 5 concludes the paper and gives directions for further research.

2. Motivation for Decentralized BPEL Navigation

In Figure 1, a common business process scenario is presented. Process participant P1 provides the depicted process model to users through a regular, centralized BPEL engine which is responsible for evaluating process control flow (depicted as drawn through arcs) that defines the order in which the activities (depicted as black filled circles 1,…,6) are executed. Process instance data is stored in variables *a* and *b*. Variable access is depicted through dashed arcs; for the examples presented in the paper no distinction is made between read- and write access. Activities 2, 4 and 5 are interaction activities (e.g. *invoke* activities) that involve remote communication with WfMS-external Web service implementations provided by process participant P2. Note that the output value *a* of interaction activity 2 is used as input for interaction activity 4. Suppose that remote Web service interactions between P1 and P2 are relatively expensive due to e.g. low communication channel bandwidth and high latency and suppose furthermore that the data contained in variable *a*

is rather large in size. If the process is executed as depicted in Figure 1 (i.e. following the established model of BPEL process execution through centralized process navigation), each invoke activity results in one remote interaction between P1 and P2. This in turn requires shipping the result of the Web service interaction with P2 through activity 2 back to the process engine at P1 and subsequently passing the same received value again back to P2 as part of the Web service interaction through activity 4.

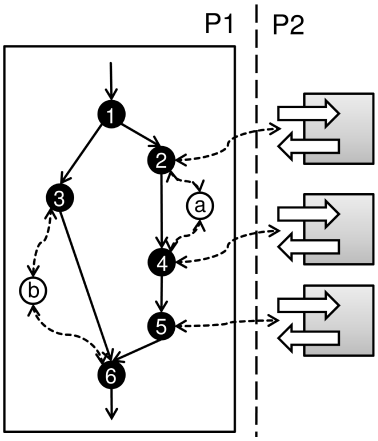

Abbildung 1: Centralized Process Execution

Clearly, the traditional BPEL model of centralized process execution and distributed Web service business logic is sub-optimal in the presented scenario, since it requires (i) a high number of remote interactions which might have an impact of the performance of process execution due to e.g. high latency of the communication channel and (ii) unnecessary shipping of potentially large amounts of data between P1 and P2.

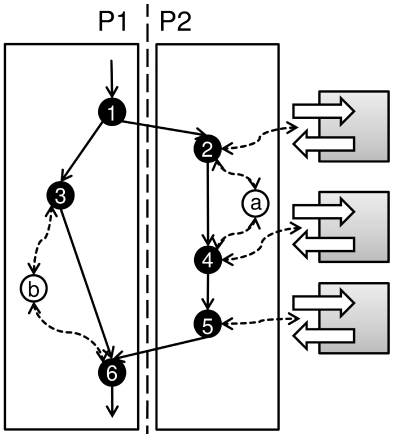

Abbildung 2: Decentralized Process Execution

In Figure 2, the same process model and Web services are presented; however, in this scenario, the execution of the BPEL process itself is split among P1 and P2 by moving the execution of the orchestration of activity implementations 2, 4 and 5 along with variable *a* from P1 to P2. As a result, after activity 1 has completed successfully, process control flow is passed to activity 2 which is provided by P2. Activities 2, 4, and 5 are then executed by P2, including the involved interactions with the WfMS-external Web services and the WfMS-internal storage of the process instance data in variable *a*. Since this interaction is done directly at the side of P2 (e.g. within the same local network), impact of communication between WfMS and service on service invocation time is reduced. Note that since there are no additional inter-dependencies between activity 3 and

activities 2, 4, 5, activity 3 can run at P1 independently of P2; furthermore, the need for shipping the data of *a* between P2 and P1 is removed. Once execution of activity 5 has finished, process control flow is passed back from P2 to P1. When both 3 and 5 have completed their execution successfully, the condition for the execution of activity 6 is fulfilled and process execution continues at P1.
While the aforementioned example scenario focuses on the performance increase resulting from shifting evaluation of process control flow as close to the actual compound business functions that are orchestrated by the respective process, other reasons can be decision drivers for this approach as well. Process models might be fragmented among a number of participants, for instance different departments of the same company for organizational reasons, or even different companies for the reasons of process outsourcing [7].

3. System Model and Architecture

To counteract the aforementioned drawbacks that result from the traditional model of centralized execution of BPEL process control flow and distributed business functions in form of Web services, we proposed another approach to execution of BPEL processes that allows for arbitrary distribution of process control flow execution among the participants of a process [8]. As discussed in Section 2, the basic goal of the approach is to shift *coordination logic* (i.e. the evaluation of process control flow) as close to the orchestrated application logic as reasonable. This is achieved through a token passing and sharing technique over remotely accessible, shared and distributed tuplespaces.

Tuplespaces have their origin in the *Linda coordination language*, defined by [9] as a parallel programming extension for traditional programming languages (e.g. C, Prolog) for the purpose of separating coordination logic from program logic. The Linda concept is built on the notion of a *tuplespace*. A user interacts with the tuplespace by storing and retrieving *tuples* (i.e. an ordered list of typed fields) via a simple interface: tuples can be (i) stored (using the *out* operation), (ii) retrieved destructively (*in*) and (iii) retrieved non-destructively (*rd*). Tuples are retrieved using a *template mechanism*, e.g. by providing values of a subset of the typed fields of the tuple to be read (associative addressing). Matching tuples are selected non-deterministically. Also, concurrent destructive reads are non-deterministic, i.e. the receiver of the tuple is chosen randomly. With respect to the degree and dimensions of decoupling they provide, Tuplespaces are conceptually similar to e.g. message-oriented middleware (see [10, 11,12] for comparisons of both technologies).

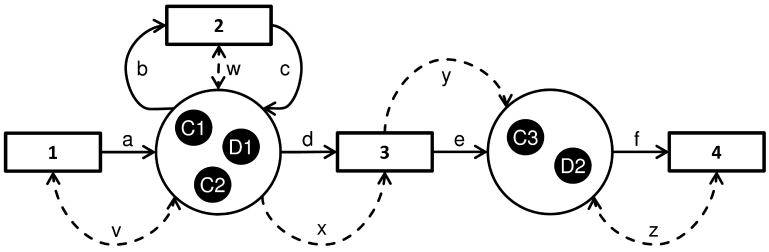

Abbildung 3: Process navigation through token passing

The overall approach to BPEL process navigation is depicted in Figure 2. Control flow is represented as control flow (*CF*) tokens (*C*1, *C*2, *C*3). *CF* tokens are encoded as tuples and communicated between the activities the process is composed of (depicted as boxes 1, 2, 3, 4) by storing them to and retrieving them from remote-accessible tuplespaces (depicted as circles); note that the term "activity" does not only refer to the Web services that the process orchestrates but also to activities whose implementations are provided by a WfMS itself, e.g. the *assign* activity. *CF*

tokens comprise information such as identifiers of the process model and process instance the *CF* token belongs to and the control flow link that the token represents. Negative process control flow, i.e. errors occurring during activity executions, is propagated between activities through so-called dead path (*DP*) tokens that are analogous to *CF* tokens in structure. *DP* tokens are necessary to facilitate *dead path elimination* [4]. A more detailed description of the Petri net-based process model is available in [8,25].

Activity implementations are realized as tuplespace clients waiting for their starting condition being fulfilled; in the remainder of the paper, activity implementations are also referred to as activity clients. The starting condition of an activity is expressed by the activity client through (i) templates describing the *CF* or *DP* tokens produced by its predecessor activity (or activities in case of activities that join multiple concurrent execution paths) and (ii) identifiers of the tuplespaces the corresponding tokens are to be consumed from. In Figure 2, *CF* token consumption operations are depicted as incoming arcs of an activity implementation. Note that these consumption operations are always synchronizing, i.e. they block until all input tokens are available and can be consumed by the respective activity client. Once the starting condition of an activity client is fulfilled, the *CF* or *DP* tokens are destructively consumed by the respective activity client and the corresponding application logic of the activity is executed. For activity executions most activity implementations require access to process instance data, i.e. variables; in the proposed model, process instance data is represented as a certain token type, called *DATA* token, that associates a certain variable of an instance of a particular process model with a value. If necessary, process instance data is consumed non-destructively or updated by the activity client during its execution (depicted as dashed incoming arcs of an activity client). Once the execution of a client's application logic has finished, corresponding *CF*, *DP* or *DATA* tokens are produced (in case of *CF* and *DP*) or updated (in case of *DATA*) depending on success or failure of activity execution; the produced tokens are stored in the tuplespace(s) the subsequent activity clients monitor for control flow tokens (depicted as outgoing arcs of activity clients).

As one can see from the aforementioned high-level description of the proposed model, tuplespaces are used as buffers for any kind of data that is communicated between the entities participating in process execution; this comprises both tuples representing process control flow (communicated via token passing) and tuples representing process instance data (communicated via token sharing). A number of reasons motivate choosing the tuplespaces as communication paradigm and platform underlying the presented approach; these are outlined in the following.

Tuplespaces have a clearly defined interface that is characterized by a number of unique features not found in alternative communication middleware such as e.g. messaging technology [13]. This interface supports operations for destructive consumption of tuples, necessary for consumption of control flow tokens, and operations for non-destructive consumption of data tuples. Furthermore, tuplespaces can be associated with *Uniform Resource Identifiers (URI)* [14] for unique identification of tuplespaces. Using a naming service, a URI can be mapped to a *Uniform Resource Locator (URL)* [15] which reflects the physical address of the machine that provides the tuplespace. The location transparency provided by tuplespaces allows clients interacting over a tuplespace to use the same communication primitives no matter whether interactions involve a remote communication with clients e.g. residing on different machines or only local interactions with clients residing on the same physical machine. Moreover, the mechanism of associative addressing enables clients to describe both their starting conditions and data access using the same language and conceptual model. In addition associative addressing frees tuplespaces from being bound to any inherent ordering of data exchanged (c.f. FIFO- or priority-based ordering of messages in message-oriented middleware systems) which facilitates e.g. co-locating *DATA* and *CF* or *DP* tokens within

the same tuplespace without preventing themselves from being consumed. Finally, tuplespaces ensure synchronized access to the tuples that are contained in a tuplespace; effectively freeing clients from any direct client-to-client interaction.

Given a system as described above, where process control flow and process instance data is communicated between process participants through remote-accessible tuplespaces, a wide spectrum of scenarios for decentralized process execution (i.e. decentralized evaluation of process control flow) can be supported using one and the same process navigation technique but different deployments of activity clients and tuplespaces; this is elaborated in the following.

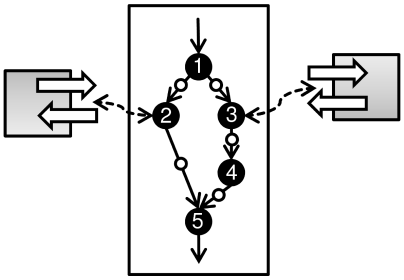

Abbildung 4: Centralized Process

On one side of the spectrum, as presented in the scenario depicted in Figure 4, a process is deployed on one single WfMS and executed using the token-passing technique outlined above. Activity clients 1,…,5 are depicted as black filled circles. Tuplespaces are depicted as smaller, unfilled circles. In this scenario all tuplespaces used for communicating control flow tokens between activity clients reside on the same physical machine. Note that for the sake of clarity access to process instance data has been omitted from the scenario descriptions and each control flow link uses one single tuplespace for communicating the corresponding control flow tokens representing that link; under the assumption that the tokens (i.e. tuples) are self-contained in the sense that they provide all the information necessary to identify a particular control link within a certain instance of a certain process model, it is however possible collate multiple tuplespaces into one single tuplespace. Since in this deployment scenario all activity implementations and tuplespaces reside on the same machine, the tuplespace interface as defined above can be used efficiently using only local interactions.

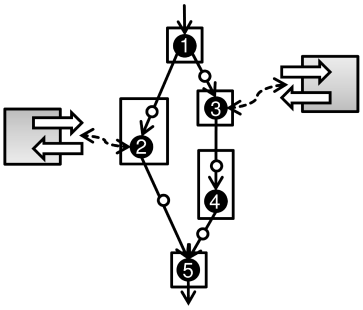

Abbildung 5: Fully decentralized process

The other side of the spectrum of decentralized process execution supported by the proposed approach is depicted in Figure 5. Here the same process has been split into five *fragments* with each fragment containing one activity client and each of those being deployed on a different machine. In general, a process fragment refers to a part of the overall process, i.e. a subset of activities and process variables. The scenario furthermore presents two possible deployment scenarios for the tuplespaces used for communicating process control flow and instance data. The communication carried out between activity 1 and 3, for instance, is conducted over a tuplespace that is located in between the two activity clients and neither resides on the same machine as activity client 1 nor

activity client 3. The interaction between activities 1 and 2, in contrast, is conducted over a tuplespace that resides at the partner providing activity client 2, hence interactions initiated by 1 (e.g. writing of positive or negative control flow tokens) are remote interactions and interactions initiated by 2 (e.g. waiting for the availability of control flow tokens and their destructive consumption) are local interactions.

Obviously, fully decentralized evaluation of process control flow as presented in Figure 5 introduces a huge amount of potentially unnecessary remote communication since each activity execution involves at least one remote interaction for the next navigation step (two in case of the tuplespace not residing on the side of either sender or receiver), so this is typically not desired. As presented in the motivating scenario (Section 2), it is often rather desired to combine multiple activity implementations into the same segment, based on the structure of the process model or the deployment of the orchestrated services.

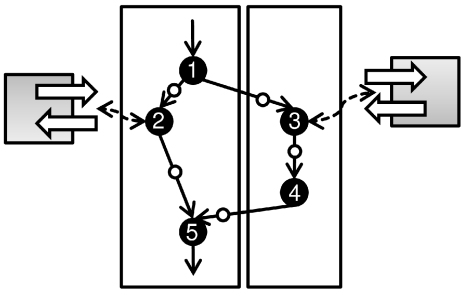

Abbildung 6: Process fragments (partially decentralized)

The way how this can be achieved is presented in Figure 6. In this case, the process is split among two participants: Activity clients 1, 2, 5 are provided by the left process participant and activity clients 3 and 4 are provided by the right participant. Similar to the deployment scenario presented in Figure 4, communication and coordination among activities 1, 2, 5 and 3, 4 (i.e. the intra-fragment communication) is achieved using local communication, while communication between 1, 3 and 4, 5 is carried out over through remote communication. Note that similar to the description in Section 3 the same tuplespace communication paradigm and primitives are used for (i) intra-fragment communication to trigger execution of subsequent activity implementations within the same fragment and to facilitate fragment-local variable access and (ii) for inter-fragment communication to pass control flow to an activity or a set of activities located in an adjacent process fragment or facilitate remote access to process instance variables.

As a summary, the spectrum of process distribution supported by the proposed method of decentralized execution of BPEL process control flow ranges from the most centralized scenario with all activity clients of a process residing on the same physical machine and both process control flow as well as process instance data tokens being exchanged over one tuplespace residing on the very same machine to processes where each activity client of a process and each tuplespace is provided on another physical machine.

3.1. Structure of Activity Clients

As described above, the infrastructure that allows for execution of processes following the proposed approach for distributed execution of BPEL processes comprises two entities: *activity clients* and *tuplespaces*. In Figure 7, the logical view upon an implementation of an activity client is depicted. An activity client comprises *coordination logic*, which defines the conditions under which a particular activity implementation can be executed and the client's *computation logic* that defines the actual application logic of the activity implementation. In tuplespace-based applications, the

coordination logic is generally provided by a *template*. A template allows defining constraints on the structure and values of a tuple through a mechanism known as *query by example* [16].

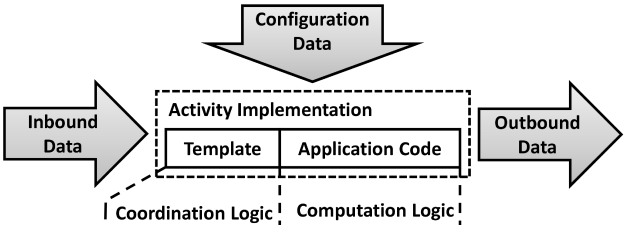

Abbildung 7: Logical View of activtiy implementations

As part of the runtime infrastructure, a parameterizable implementation of each BPEL activity's application logic is provided. In case of e.g. the BPEL *assign* activity, this comprises the consumption of the input variables, the application of the assign expression and the update of the value of the output variable. Activity clients are parameterizable through external configuration data in such a way, that the URIs of the tuplespaces containing inbound and outbound control flow tokens and data tuples and corresponding templates describing the tuples to be consumed can be defined. Since activity clients will typically be deployed on a number of different physical machines, remote configuration (i.e. parameterization) of activity clients must be supported.
For facilitating efficient communication between activity clients deployed on one physical machine for intra-segment communication as well as remote communication between activity clients deployed on different machines for inter-segment communication, two communication protocols for the interaction between activity clients and the provided tuplespace are supported.

4. Related Work

In [18], a motivation for distributed execution of workflows is presented and a description of the abstract components which form a distributed workflow application and their relations is presented. In addition, the authors provide an overview of existing distributed WfMS. In [19], the authors introduce a workflow management system architecture for supporting large-scale distributed applications, called *Mentor*, which is based on TP monitors and object request brokers. Decentralized workflow execution in Mentor is achieved by partitioning a workflow based on activity and state charts into a set of sub-workflows which are then enacted by a number of distributed workflow engines that are synchronized using Mentor; hence, in this approach, the minimal unit of fragmentation is a (sub-) process and not an individual activity as in the approach presented in this paper. In [20], $Adept^{Distribution}$, a similar approach to workflow partitioning and decentralized execution is presented that supports dynamic assignment of workflow partitions to so-called execution servers. Another approach to enacting workflows in a mobile, pervasive environment which relies on passing control flow between distributed BPEL engines is presented in [21]. However, no insight is given as to which extend this approach supports BPEL's advanced features such as fault- and compensation-handling and dead path elimination. In [22], the distributed workflow engine *OSIRIS* is presented which is based on the concept of a *hyper-database*. The hyper-database is a distributed peer-to-peer database system that offers a number of global services with ACID transactional semantics and is realized by a hyper-database layer on each node participating in the process execution. A distributed publish-subscribe infrastructure is used for keeping data relevant for process instance execution on participating nodes in sync. In [7], an approach to fragmenting BPEL process for the purpose of business process outsourcing is presented which involves (i) a local BPEL workflow engine at each partner participating in the workflow execution and (ii) a centralized coordinator in between partners that is necessary for supporting certain BPEL constructs such as scopes and while loops split across multiple partners. A

BPEL process model is manually split by a user in a number of fragments and a corresponding BPEL process (along with necessary deployment information) is created for each fragment. Exchange of process instance data and passing control flow between fragments is realized by introducing additional *invoke* and *receive* activities in each process fragments, for sending (i.e. pushing) a message containing instance data to be passed to another fragment on one side and receiving potentially multiple messages containing instance data from different partners (through so-called *receiving flows*) on the other side. While this method retains standards-compliance by reusing existing Web service standards (e.g. WS-Coordination) it requires a centralized coordinator in cases where loops or scopes are split among partners. In addition, since process instance data is communicated between process participants through a push mechanism, complex data flow analysis is required on the sender's side to determine the data that needs to be communicated. In [23], a cost model and algorithm for automatically splitting BPEL processes among a number of participating BPEL workflow engines is presented that is based on program dependence graphs. Similar work focused on more dynamic, mobile scenarios is presented in [24].

5. Conclusions and Future Work

In the paper we have presented an infrastructure and conceptual model for decentralized, distributed execution of business processes in BPEL that allows for arbitrarily distributed evaluation of process control flow. We have motivated the need for distributed evaluation of process control flow through an introductory example. The supported spectrum of process distributed has been discussed and the basic building blocks of the system architecture have been described. In contrast to existing models and techniques the approach as proposed in the paper allows for arbitrarily fragmented processes while using the same mechanisms (but different transport protocols) for evaluation of process control flow at the side of one process participant and between process participants. A tuplespace prototype that provides the necessary functionality for communicating process control flow between local activity clients has been implemented. While the broad spectrum of process fragmentation supported by the proposed system, allows for a wide range of optimizations depending on a user's requirements, it requires the definition of an corresponding algorithm for automatic creation of those fragments. The definition of such an algorithm, based on process models, infrastructure descriptions and user requirements and the extension of the current prototype to support decentralized process deployment is considered future work.

6. References

[1] KRAFZIG, D., BANKE, K., SLAMA, D.: Enterprise SOA. Service Oriented Architecture Best Practices. Prentice Hall International (December 2004)

[2] WEERAWARANA, S., CURBERA, F., LEYMANN, F., STOREY, T., FERGUSON, D.F.: Web Services Platform Architecture: SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging and More. Prentice Hall PTR Upper Saddle River, NJ, USA (2005)

[3] ALONSO, G., CASATI, F., KUNO, H., MACHIRAJU, V.: Web Services: Concepts, Architectures and Applications. Springer Verlag (2004)

[4] ARKIN, A., ET AL.: WS-BPEL: Web Services Business Process Execution Language Version 2.0. OASIS Open, January (2007)

[5] LEYMANN, F.: Web Services: Distributed Applications without Limits, in: Business, Technology and Web, Leipzig (2003)

[6] LEYMANN, F., ROLLER, D.: Production Workflow: Concepts and Techniques. Prentice Hall PTR Upper Saddle River, NJ, USA (1999)

[7] KHALAF, R., LEYMANN, F.: Role-based Decomposition of Business Processes Using BPEL, in: International Conference of Web Services, ICWS 770–780

[8] WUTKE, D., MARTIN, D., LEYMANN, F.: Model and infrastructure for decentralized workflow enactment, in: 23rd ACM Symposium on Applied Computing (SAC2008) (2008)

[9] GELERNTER, D.: Generative Communication in Linda, in: ACM Transactions on Programming Languages and Systems 7(1) (1985) 80–112
[10] LEYMANN, F.: Space-based Computing and Semantics: A Web Service Purist's Point-Of-View (2006)
[11] FENSEL, D., EVA KÜHN, LEYMANN, F., TOLKSDORF, R.: Queues Are Spaces - Yet Still Both Are Not The Same? (2007), http://www.spacebasedcomputing.org/fileadmin/files/SBC-QueuesAreSpaces_20070511.pdf
[12] MARTIN, D., WUTKE, D., SCHEIBLER, T., LEYMANN, F.: An EAI Pattern-based Comparison of Spaces and Messaging. In: EDOC '07: Proceedings of the 11th IEEE International Enterprise Distributed Object Computing Conference, Washington, DC, USA, IEEE Computer Society (2007) 511
[13] BLAKELY, B., HARRIS, H., RHYS, L.: Messaging and Queueing Using the MQI: Concepts & Analysis, Design & Development. McGraw-Hill (1995)
[14] BERNERS-LEE, T., FIELDING, R., MASINTER, L.: RFC 3986: Uniform Resource Identifier (URI): Generic Syntax. The Internet Society (2005)
[15] BERNERS-LEE, T., MASINTER, L., MCCAHILL, M.: RFC 1738: Uniform Resource Locator (URL). The Internet Society (1994)
[16] ZLOOF, M.: Query-by-Example: A Data Base Language, in: IBM Systems Journal 16(4) (1977) 324–343
[17] CLARK, J., DEROSE, S.: XML Path Language (XPath). W3C Recommendation (November 1999)
[18] JABLONSKI, S., SCHAMBURGER, R., HAHN, C., HORN, S., LAY, R., NEEB, J., SCHLUNDT, M.: A comprehensive investigation of distribution in the context of workflow management, in: International Conference on Parallel and Distributed Systems ICPADS, Kyongju City, Korea (2001)
[19] MUTH, P., WODTKE, D., WEISSENFELS, J., DITTRICH, A., WEIKUM, G.: From Centralized Workflow Specification to Distributed Workflow Execution, in: Journal of Intelligent Information Systems 10(2) (1998) 159–184
[20] BAUER, T., DADAM, P.: Efficient Distributed Workflow Management Based on Variable Server Assignments, in: International Conference on Advanced Information Systems Engineering (CAiSE'00), Stockholm (2000) 94–109
[21] MONTAGUT, F., MOLVA, R.: Enabling Pervasive Execution of Workflows, in: 1st IEEE International Conference on Collaborative Computing: Networking, Applications and Worksharing, CollaborateCom (2005)
[22] SCHULER, C., WEBER, R., SCHULDT, H., SCHEK, H.J.: Scalable Peer-to-Peer Process Management - The OSIRIS Approach, in: IEEE International Conference on Web Services 2004 26–34
[23] NANDA, M.G., CHANDRA, S., SARKAR, V.: Decentralizing Execution of Composite Web Services, in: 19th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications (2004) 170–187
[24] BARESI, L., MAURINO, A., MODAFFERI, S.: Workflow Partitioning in Mobile Information Systems, in: IFIP TC8 Working Conference on Mobile Information Systems, Oslo, Norway (2004) 93–106
[25] MARTIN, D. , WUTKE, D., LEYMANN, F.: A Novel Approach to Decentralized Workflow Enactment. 12th IEEE International EDOC Conference, Munich, Germany (2008)