A Method for Partitioning BPEL Processes for Decentralized Execution*

Daniel Wutke, Daniel Martin, and Frank Leymann

Institute of Architecture of Application Systems
University of Stuttgart
Universitätsstrasse 38, 70569 Stuttgart, Germany
{wutke,martin,leymann}@iaas.uni-stuttgart.de
http://www.iaas.uni-stuttgart.de

Abstract Service orchestrations are a common means to compose individual services to either higher-level services or potentially complex composite applications. The Web Service Business Process Execution Language (WS-BPEL) is an example for a language that allows for defining automatically executable orchestrations of Web services. As of today, BPEL process are typically executed in a centralized manner; the process model is deployed on a single workflow management system which, during process instance execution, interprets the process definition and interacts with the orchestrated Web services on behalf of the user. In previous work, we have presented an approach which enables decentralized execution of BPEL processes based on a decentralized process model and supporting runtime infrastructure. In this paper we describe a method for automatic splitting of a process among the partners participating in its execution, referred to as *process partitioning*.

Key words: Process partitioning, decentralized process enactment, BPEL

1 Introduction

One of the key aspects of the *Service-oriented Architecture (SOA)* are service compositions following the so-called *two-level programming* paradigm where individual reusable services are composed into high-level services or potentially complex service orchestrations which can executed automatically using workflow management systems (WfMS). The means for defining the "wiring" between the compound services is provided by workflow definition languages such as the *Web Service Business Process Execution Language (WS-BPEL)* [1]. As of today, the automatic execution of a BPEL process typically comprises the deployment of the process model to a single WfMS and – after process instantiation triggered by incoming messages sent by clients to the WfMS – continuous evaluation of (i) the process' control flow defined in the process model and (ii) the current state of

* This work is supported by the EU funded project *TripCom* (FP6-027324), http://www.tripcom.org.

the process' instance data by the WfMS's *navigator* component; hence we refer to process navigation being a *centralized* process.

However, a number of reasons, ranging from outsourcing of process fragments to runtime performance optimizations without the need for process model changes, motivate the need for a decentralized execution environment for BPEL processes. Hence, in previous work [2,3,4], we have presented an approach that allows for (nearly) arbitrary process splitting by enabling distributed navigation among the partners participating in a process' execution. Following this approach, once can realized different deployment of the same process model within the spectrum from ranging from centralized execution to fully decentralized execution where each "step" (i.e. activity) being carried out by the process is executed by a different process participant. Finding an "adequate" distribution, also referred to as *partitioning*, of the process is dependent on a number of influential factors. Subject of this paper is the presentation and discussion of these factors and a high-level description of an approach to the process partitioning problem that addresses each of the identified influential factors.

The remainder of the paper is structured as follows. In Section 2 the EWFN process model provides the basis for the proposed approach is described to the extend necessary for the discussion of the partitioning procedure. Based on this foundation, Section 3 introduces the general idea of process partitioning, discusses various parameters that influence process partitioning and outlines a procedure that addresses each of the defined partitioning parameters. In Section 4 a brief overview of a few related approaches which are either particularly relevant to the problem discussed in the paper due to either addressing the concrete problem of partitioning BPEL processes or similar parameters influencing process partitioning are presented.

2 Decentralized Enactment of BPEL Processes

Coordinating a number of distributed clients, where each of those clients realizes a defined part of an overall process requires communication of both *process control flow* and *process instance data* among the clients participating in the process' execution. In this context, control flow refers to the individual client's execution being started according to the order defined in the process model; the term instance data characterizes both data being "visible" in process models such as BPEL variables, partner links (which can be source or destination of assignment operations) or correlation sets as well as "invisible" instance data such as the state of a scope or the state of incoming message activities. While this information is provided in the WS-BPEL specification through a description of the language's operational semantics, this description is – due to its informal textual nature – neither suitable for automatic execution by WfMSs nor may it serve as input for process partitioning.

As a result, the formalism of *Executable Workflow Networks (EWFN)* has been developed on the basis of colored, non-hierarchical Petri nets [5] and Boolean networks [6]; it allows for explicitly describing the data communicated

during process instance execution using the communication primitives of the the Linda Tuplespace model [7] – *read* for non-destructive and *take* for destructive consumption of data, *write* for production of data – plus a number of extensions that address process execution-specific requirements such as the *sync* operation for synchronizing join operations [8].

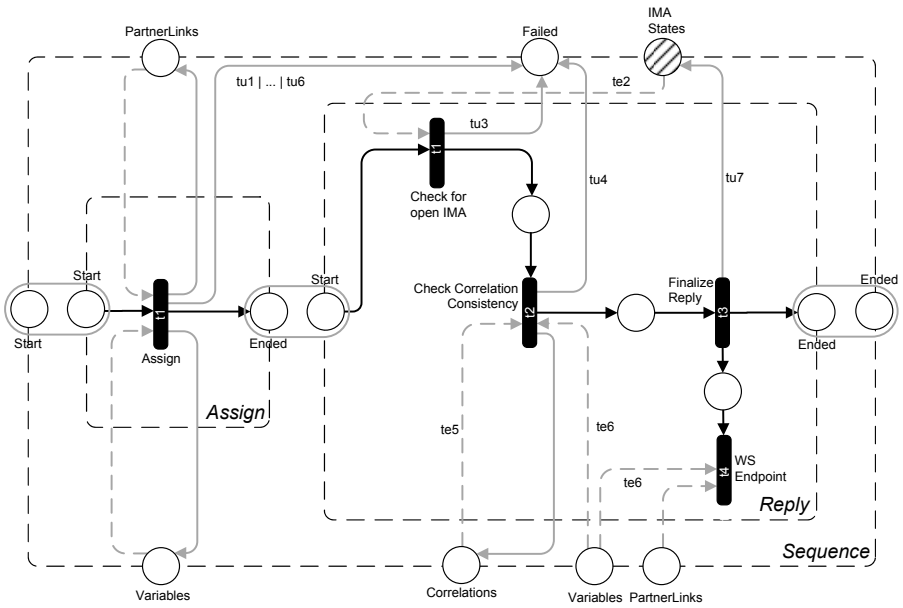

Figure 1. Example of the EWFN representation of a BPEL SEQUENCE activity with two contained `assign` and `reply` activities.

Figure 1 depicts the EWFN representation of a BPEL `sequence` activity with two contained activities. Activities are depicted as dashed rectangles and comprise (similar to Petri nets) transitions and places. Following the EWFN formalism, transitions represent a piece of application logic and carry out the actual processing. Transitions coordinate themselves by consuming tokens from and producing tokens to places which provide passive buffers for tokens; the tokens are self-contained in the sense that they provide enough information to uniquely identify each token communicated in an EWFN. On the level of the infrastructure supporting execution of EWFNs, places are realized by tuplespaces, transitions by tuplespace clients. The arcs between transitions and places represent the individual operations supported by the tuplespace interface and may be annotated with a weight representing the operation cost, resulting e.g. from the volume of the data being communicated as part of that operation.

As an example, transition $t1$ of the `assign` activity represents an assignment of a value to either a BPEL variable or a partner link. To represent process control flow $t1$ is activated by a token becoming available in its *Start* place; once $t1$ has finished its execution is signals its successful completion to the

subsequent activity by producing a corresponding token to its *Ended* activity. Consuming and producing process control flow information is depicted as directed black arcs between places and transitions and transitions and places respectively. During execution of their application logic, transitions may consume and produce further tokens representing instance data; in case of the depicted `assign` activity this might include the modification of the value of a variable or the endpoint reference assigned to a partner link. Similar to BPEL activities, EWFNs can be nested as presented in the example with the `sequence` activity surrounding the `assign` and `receive` activities. The operational semantics of the `sequence` activity is defined as each contained activity becoming ready to execute once its preceding activity has completed its execution. In the EWFN this is represented by collapsing the *Ended* place of the `assign` activity and the *Start* place of the `receive` activity into one place.

3 Process Partitioning

The term *process partitioning* refers to the procedure of assigning information about the partner the corresponding node of the EWFN is executed by during instance runtime to each transition and place of an EWFN. This process is influenced by a number of parameters which can be classified in three groups.

Process model As outlined before, the EWFN of a BPEL process provides a formal description of (i) the steps carried out during process instance execution and (ii) the data being communicated along the way. As a result it is one of the major parameters of the process partitioning procedure.

Service infrastructure Through BPEL's interaction activities (e.g. `invoke` and RECEIVE) the BPEL process may interact with Web service clients and the Web services it orchestrates. As the partners providing a service used by the process are in any event process participants they are potentially suitable candidates for executing a part of the processes orchestration logic as well.

Organizational factors Organizational factors reflect parameters that are not necessarily a result of the process structure or its service landscape, but are defined manually by users. It might e.g. be desired to manually define the partition of a certain place that contains BPEL variable data for data ownership reasons.

The proposed process partitioning approach comprises three phases and is an extension of the procedure presented in [9] in the sense that it also relies on the notion of different kinds of nodes – *fixed*, *heavy*, and *light* – whose partition assignment is addressed in consecutive phases of the partitioning algorithm as depicted in Table 1; once a partition of a been determined in on of the phases the node is not considered in the further phases of the algorithm.

Fixed nodes represent nodes with partitioning information defined a priori by manual user input and reflect organizational partitioning parameters. To allow for maximum flexibility, each node of an EWFN – transitions and places – can become a fixed node. Interaction activities, i.e. those points of a BPEL process

Phase	1. Fixed Nodes	2. Heavy Nodes	3. Light Nodes
Examined Objects	Arbitrary nodes	Interaction activities: `invoke`, `receive`, `pick`, `reply`	Non-interaction activities; instance data
Partitioning Method	Manual assignment by user; rules	Automatic *service discovery* and *service selection*; rules	Graph partitioning applied to the process' EWFN

Table 1. Phases of the proposed method to BPEL process partitioning.

where interaction with its service landscape occurs, are referred to as *heavy nodes*. Their partitioning information is determined automatically using means for *service discovery* (based on the service's functional characteristics through its WSDL description) and *service selection* (based on the service's non-functional properties such as service invocation cost, service response time, etc. reflected through WS-Policy descriptions). In addition, partitioning information of *heavy nodes* might be dependent on deployment information (e.g. for defining on which endpoint the process can receive incoming messages) or other *heavy nodes* (e.g. in case of `receive-reply` pairs to support synchronous Web service bindings). *Light nodes* reflect non-interaction activities as well as process instance data for which no partitioning information has been defined until this point. Their partitioning is performed by migrating them to the partition of adjacent *fixed* or *heavy nodes* as defined in the process' EWFN. Since an EWFN is a directed weighted graph and the problem is a variant of the well-known *graph partitioning* problem with the optimization criteria of minimizing the cost of inter-partition interactions, existing optimization algorithms such as *Simulated Annealing* [10] are used to realize this phase of the partitioning algorithm.

4 Related Work

In [11], the authors introduce a workflow system architecture for supporting large-scale distributed applications called *Mentor* based on TP monitors and object request brokers. Decentralized workflow execution in Mentor is achieved by rule-based partitioning of 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. In [12], a BPEL process model is manually split by a user (e.g. for reasons of process outsourcing) in a number of fragments and a corresponding BPEL process (along with necessary deployment information) is created for each fragment. The BPEL processes are then deployed and executed at the partners participating in the process' execution. For supporting BPEL's `scope` and `while` activities, a central coordinator is required. In [9] a similar approach to distributed execution of BPEL processes is presented supporting automatic process partitioning based on an analysis of a program dependence graph generated for the process and a corresponding cost model.

5 Conclusions

In this paper we have outlined a process model that enables decentralized execution of BPEL processes and allows for nearly arbitrarily fragmented process execution. Thereby we have stressed the need for an algorithm for defining process partitions based on a number of influential factors and have presented a high-level overview of the proposed algorithm and how it addresses the individual process partitioning parameters.

References

1. Organization for the Advancement of Structured Information Standards: Web Services Business Process Execution Language Version 2.0 – OASIS Standard (March 2007)
2. Wutke, D., Martin, D., Leymann, F.: Model and Infrastructure for Decentralized Workflow Enactment. In: SAC '08: Proceedings of the 2008 ACM Symposium on Applied Computing, New York, NY, USA, ACM (2008) 90–94
3. Daniel Martin and Daniel Wutke and Frank Leymann: EWFN – a Petri net dialect for tuplespace-based workflow enactment. Volume 380 of CEUR Workshop Proceedings., CEUR-WS.org (September 2008) 7–14
4. Martin, D., Wutke, D., Leymann, F.: A novel approach to decentralized workflow enactment. Enterprise Distributed Object Computing Conference, 2008. EDOC '08. 12th International IEEE (Sept. 2008) 127–136
5. Jensen, K.: Coloured Petri Nets, Vol. 1: Basic Concepts. EATCS Monographs on Theoretical Computer Science. Berlin, Heidelberg, New York: Springer-Verlag (1992)
6. Langner, P., Schneider, C., Wehler, J.: Prozessmodellierung mit ereignisgesteuerten Prozessketten (EPKs) und Petri-Netzen. Wirtschaftsinformatik **39**(5) (1997) 479–489
7. Gelernter, D.: Generative Communication in Linda. ACM Transactions on Programming Languages and Systems **7** (1985) 80–112
8. Martin, D., Wutke, D., Leymann, F.: Synchronizing control flow in a tuplespace-based, distributed workflow management system. In: ICEC '08: Proceedings of the 10th international conference on Electronic commerce, New York, NY, USA, ACM (2008) 1–9
9. Nanda, M.G., Chandra, S., Sarkar, V.: Decentralizing Execution of Composite Web Services. Proceedings of the 19th Annual ACM SIGPLAN Conference on Object-oriented Programming, Systems,Languages, and Applications (2004) 170–187
10. Kirkpatrick, S., Gelatt, C., Vecchi, M.: Optimization by Simulated Annealing. Science **220**(4598) (1983) 671–680
11. Muth, P., Wodtke, D., Weissenfels, J., Dittrich, A., Weikum, G.: From Centralized Workflow Specification to Distributed Workflow Execution. Journal of Intelligent Information Systems **10**(2) (1998) 159–184
12. Khalaf, R., Leymann, F.: Role-based decomposition of business processes using bpel. In: ICWS '06: Proceedings of the IEEE International Conference on Web Services, Washington, DC, USA, IEEE Computer Society (2006) 770–780