Useful Patterns for BPEL Developers

Darko Andročec, Dragutin Kermek
Faculty of Organization and Informatics
University of Zagreb
Pavlinska 2, 42000 Varaždin, Croatia
{darko.androcec, dragutin.kermek}@foi.hr

Abstract. *Modern enterprises use service-oriented architecture to accommodate business agility. One of the most important SOA standards is BPEL, because it provides a means to specify business processes and their interaction. In this paper we examine patterns that are related to BPEL or can be implemented in BPEL. The most important identified pattern types are SOA patterns, workflow patterns and integration patterns. We analyze these aforementioned sets of patterns, sort out and briefly describe patterns used for web service composition and patterns that can be implemented in BPEL. The main aim of this paper is to list existing patterns that are relevant to BPEL developers.*

Keywords. Pattern, service oriented, BPEL, workflow, integration

1 Introduction

To accommodate business agility, modern enterprises can use web services to form a set of autonomous loosely coupled business processes [14]. The purpose of the service-oriented architecture (SOA) is to address the requirements of loosely coupled and protocol-independent distributed computing. The main resources of SOA are services which provide business functionality and have independent state and context. Services are described using standard definition language and have a published interface. SOA is a design philosophy which is independent of any specific technology. Service is a pair of a service interface (the identity of a service and its invocation) and a service implementation. All functions in a SOA are autonomous services. Nowadays web services are the preferred implementation technology for service-oriented architecture. BPEL is a process-oriented language for web service composition.

We examine patterns which are useful for web service compositions developers (patterns that are related to BPEL or can be implemented in BPEL). This paper proceeds as follows. Firstly, in Section 2, we present a brief overview of the BPEL, a language used for web service composition. Section 3 shows related work. In the next section we analyze BPEL related patterns, sort out and briefly describe patterns used for web service composition and patterns that can be implemented in BPEL. Finally, our conclusions are presented in the last section.

2 BPEL

The logic of the interactions between a service and its environment in BPEL [13] is described as a composition of actions (receive, send etc.). There are also control-flow constructs corresponding to parallel, sequential and conditional execution, event and exception handling and compensation. Data manipulation is done using lexically scoped variables. BPEL supports the description of the behavior of a class of services and execution order of a set of activities. There are two categories of activities in BPEL. Basic activities correspond to atomic actions (invoke, receive, reply, wait, assign, etc.). Structured activities deal with constraints on a set of activities contained within them (sequence, flow, switch, while, scope). Control links support the definition of precedence, synchronization and conditional dependencies together with notions of join and transition condition. BPEL also defines events, fault and compensation handlers.

OASIS Web Services Business Process Execution Language Version 2.0 specification is given in [11] and [12]. BPEL is a language for formally describing business processes and their

interactions; it is a language for business process orchestration based on web services. A BPEL process element is an outermost container where you can declare relationships to external partners, process data, handlers and the activities to be executed [11].
Typed variables hold the data of the state of a BPEL business process during runtime. The major building blocks of BPEL processes are structured and basic activities.

The receive activity is used for receiving messages from an external partner. It specifies the partner link, operation of the partners, and a variable or a set of variables that holds the requested data. One receive activity can have an associated reply activity that can reply normal or faulted data. A web service provided by a partner is called by the invoke activity. The sequence activity is used to define a collection of activities which are executed sequentially. The if-else activity enables to select exactly one branch of the activity from a set of choices. BPEL also offers three activities that allow repeated execution: while, repeatUntil and forEach activity. The flow BPEL activity is used to execute business logic in parallel. You can add a link activity when needed. It introduces a dependency that the target activity of the link will only be executed if the source activity of the link has completed. The assign activity is used for data manipulation and it contains one or more copy operations. The validate activity and the validate attribute are used for data validation. BPEL has fault handlers that can be attached to a scope, a process or on an invoke activity.

A business process in BPEL can be structured into a hierarchy of nested scopes. A scope begins with initialization and finishes successfully or unsuccessfully (internal or external fault). The process and each scope may define event handlers for processing web service request messages arriving in parallel to the primary activity. The wait activity enables delayed execution of the business logic, and exit activity immediately ends all currently running activities. The empty activity is used when no action is to be taken.

3 Related work

In this section we present related work about pattern usage in service oriented architecture and BPEL. Hohpe [9] concluded that SOA implementation exposes new and unfamiliar programming models which require a different way of thinking and also new guidance and patterns. Zdun, Hentrich, and Dustdar [23] proposed using patterns and patterns primitives for modeling process-driven and service-oriented architectures. Their main contribution is a modeling concept based on pattern primitives (precisely specified modeling elements) specified as extensions of UML2. They used OCL (Object Constraint Language) to specify constraints and precise semantics for the pattern primitives. Zdun [24] showed how to use patterns for design of service-oriented middleware. He also presented a novel semantic lookup service concept where each peer (service) provides its semantic metadata to federation's lookup service. Aoyama and Mori [1] presented design method of aSOA (asynchronous service-oriented architecture) based on patterns. Authors proposed model-driven design methodology for aSOA, created a set of aSOA patterns and implemented prototype on the Apache Axis. Chengjun [2] tried to apply pattern oriented software engineering to web service development and used identified patterns as the means to express the results of different development phases.

Patterns for business object model integration in process-driven and service-oriented architectures are shown in [7]. When multiple business object models must be integrated, many data integration issues might arise, so Hentrich and Zdun [7] present patterns that address integration issues. Gomma et al. [6] described the concept of software adaptation patterns and their use in service oriented architectures. These patterns define how a set of components of architecture pattern cooperate to change the software configuration. Van Lessen, Nitzsche and Leymann [22] proposed a way to formalize message exchange pattern (conversational contracts between a service consumer and service provider) using BPELlight (extension of BPEL 2.0) through introducing of a new abstract BPEL profile. An approach based on mediator patterns to generate executable mediators and connect partially compatible services is proposed in [10]. A heuristic technique for identifying protocol mismatches and selecting appropriate patterns with their corresponding BPEL templates is presented as well in the aforementioned work.

4 BPEL related patterns

After publishing of the book [5] the interest of software developers for design patterns was increased. The design patterns are descriptions of communicating objects and classes that are customized to solve a general design problem in a particular context [5]. A pattern has the four essential elements: pattern name, problem, solution and consequences. In this section we will examine patterns that are useful for BPEL developers. The most important related pattern types are SOA patterns, workflow patterns and integration patterns.

4.1 SOA patterns

The most comprehensive review of SOA design patterns is given in the book [4]. "Each SOA design pattern provides a design solution in support of successfully applying service orientation and establishing a quality service-oriented architecture."[14] Erl [3,4] broke down SOA architectural model into the following types: service architecture, service composition architecture, service inventory architecture, service-oriented enterprise architecture. He defined four basic categories of SOA patterns: inventory boundary patterns, inventory structure patterns, service design patterns, service composition and communication patterns. Inventory boundary patterns determine the appropriate scope of a service inventory. Inventory structure patterns are applied to ensure a consistent structure in support of service orientation. Patterns for service design are applied at the service architecture level to solve numerous challenges of service-orientation design principles. Patterns for service composition and communication address composition-related design issues.

We will now mention the most important design patterns from Erl's book [4] for BPEL developers (patterns that are related to BPEL or can be implemented in BPEL). The first important pattern is Process Centralization [4]. A problem can arise when business process logic is distributed across independent service implementations because it is hard to extend and evolve. The solution is to deploy and govern logic representing numerous business processes from a central location. Middleware platforms with BPEL support generally provide the necessary orchestration technologies to apply this pattern. A pattern Compensating Service Transaction can also be applied with help of BPEL features. When there are uncontrolled runtime exceptions in a service composition, wrapping the composition in an atomic transaction can negatively affect performance and scalability. Solution is introducing of compensating routines to allow runtime exceptions resolve and reduce resource locking and memory consumption. Another important pattern from book [4] related to BPEL is a compound pattern Orchestration. It is dedicated to the effective maintenance and execution of parent business process logic through support of sophisticated and complex service composition logic that can result in long-running runtime activities. This compound pattern consists of the application of Process Abstraction, State Repository, Process Centralization, and Compensating Service Transaction patterns. Orchestration engine can support industrial standards including BPEL.

4.2 Workflow patterns

Russel proposed workflow patterns in his doctoral thesis [19] and papers written in co-operation with other authors [15, 16, 17, 18]. Workflow patterns identify comprehensive workflow functionality and provide the basis for a comparison of available workflow management systems.

There are four perspectives of workflow patterns [19]: control-flow, data, resource and exception handling perspective. The control-flow perspective describes the structure of a process model (implementation of its constituent activities and the interconnections between them). The data perspective describes definition and utilization of data elements. The resource perspective includes the overall organizational context of process execution. The exception handling perspective deals with the specification of exception handling strategies for expected or unexpected events during execution.

Table 1. List of workflow patterns supported by BPEL

Perspective of workflow patterns	**Direct support**	**Partial support**
Control-flow	*Sequence, Parallel Split, Synchronization, Exclusive Choice, Simple Merge, Multi-Choice, Structured Synchronizing Merge, Implicit Termination, Multiple Instances without Synchronization, Deferred Choice, Cancel Activity, Cancel Case, Structured Loop, Persistent Trigger, Acyclic Synchronizing Merge, Critical Section, Interleaved Routing*	*Interleaved Parallel, Cancel Region, Thread Merge, Thread Split*
Data	*Scope Data, Case Data, Environment Data, Data Interaction – Task to Task, Data Interaction – Task to Environment – Push-Oriented, Data Interaction – Environment to Task – Pull-Oriented, Data Transfer by Value – Incoming, Data Transfer by Value – Outgoing, Data Transfer by Reference – Unlocked, Task Precondition – Data Value, Event-based Task Trigger, Data-based Routing*	*Task Data, Data Interaction – Case to Case, Data Interaction – Environment to Task – Push-Oriented, Data Interaction – Task to Environment – Pull-Oriented, Data Transfer by Reference – With Lock, Task Precondition – Data Existence, Data-based Task Trigger*

BPEL directly supports some workflow patterns described in [19]. Of the 43 published workflow patterns from control-flow perspective [17], BPEL directly support 17 patterns and partially 4. From catalogue of workflow data patterns [16] BPEL supports 12 patterns and partially support another 7 ones. These patterns are listed in Table 1. BPEL does not provide direct support for resources in business processes based on web services, so BPEL does not support workflow resource patterns. Some workflow exception handling patterns can be implemented in BPEL [18].

4.2 Integration patterns

Patterns can be used to solve a variety of integration problems. The book [8] proposed enterprise integration patterns which help integration architects and developers to design and implement integration solutions more rapidly and reliably. In BPEL we create the service relationships and create containers for the input and output messages of the services. Containers are very similar to Datatype Channel enterprise integration pattern for the WSDL message types. Datatype Channel [8] is used to enable an application to send a data item such that the receiver will know how to process it. Separate Datatype Channel is used for each data type, so that all data on a particular channel is of the same type.

Container can also be considered as an extended version of the Shared Database pattern. Shared Database [8] answers to the question how to integrate multiple applications so they can work together and exchange information. A solution is using of a central data store that all of the applications share. BPEL is also an implementation of Process Manager pattern because it manages and correlates the flow of messages between services. Process Manager [8] tries to solve a problem of routing a message through multiple processing steps when the required steps may not be known at design-time and may not be sequential. Process Manager is a central processing unit which maintains the state of the sequence and determines the next processing step based on intermediate results. BPEL is a standardized intermediate language between the process modeling tools and the Process Manager Engine.

Microsoft published another book about integration patterns [21]. Process Integration is a pattern from this book which is connected to BPEL (it describes process models). It solves the problem of execution coordination of a business function that spans multiple applications. A solution is definition of a business process model that describes the individual steps of the complex business function. Separate process manager component must be created. This component interprets concurrent instances of the business model and interacts with the existing applications to perform the process steps.

5 Conclusion

In this paper we analyze BPEL related patterns and conclude that the most important sets of patterns are SOA patterns, workflow patterns and integration patterns. We analyze the aforementioned sets of patterns, sorted out and briefly describe patterns used for web service composition and patterns that can be implemented in BPEL.

Related SOA patterns propose how to deploy and govern logic representing numerous business processes from a central location, introduce compensating routines to allow runtime exceptions resolving and reduce resource locking and memory consumption, and provide the effective maintenance and execution of parent business process logic through support of sophisticated and complex service composition logic. BPEL directly supports 45 and partially supports 11 patterns of the 218 identified workflow patterns (control-flow, data and exception handling perspective). Workflow resource patterns are not supported in BPEL, because BPEL does not provide direct support for resources in business processes based on web services. Related integration patterns enable an application to exchange information, answer to the question how to integrate multiple applications so they can work together, manage and correlate the flow of messages between services, and solve the problem of execution coordination. Patterns mentioned in this work are very useful for BPEL developers because they provide proven solutions to web service composition problems. This is ongoing research and we plan to analyze other types of useful patterns for BPEL developers in our future work.

References

[1] Aoyama M, Mori A: **A Unified Design Method of Asynchronous Service-Oriented Architecture Based on the Models and Patterns of Asynchronous Message Exchanges**, Proceedings of the 2008 IEEE International Conference on Web Services (ICWS '08), Washington DC, USA, 2008, pp. 537-544.

[2] Chengjun W: **Applying Pattern Oriented Software Engineering to Web Service Development**, Proceedings of the 2008 International Seminar on Future Information Technology and Management Engineering (FITME '08), Washington DC, USA, 2008, pp. 214-217.

[3] Erl T: **Introducing SOA Design Patterns**, http://soa.sys-con.com/node/645271, Accessed:15th September 2011.

[4] Erl T: **SOA Design Patterns**, Prentice Hall – Pearson Education, Boston, USA, 2009.

[5] Gamma E, Helm R, Johnson R, Vlissides J: **Design Patterns – Elements of Reusable Object-Oriented Software**. Addison-Wesley Professional, Boston, USA, 1995.

[6] Gomaa H, Hashimoto K, Kim M, Malek S, Menasce D A: **Software Adaptation Patterns for Service-Oriented Architectures**, Proceedings of the 2010 ACM Symposium on Applied Computing (SAC '10), New York, USA, 2010, pp. 462-469.

[7] Hentrich C, Zdun U: **Patterns for Business Object Model Integration in Process-Driven and Service-Oriented Architectures**, Proceedings of the 2006 conference on Pattern

languages of programs (PLoP '06), New York, USA, 2006, pp. 1-14.

[8] Hohpe G, Woolf B: **Enterprise integration patterns – Designing, Building and Deploying Messaging Solutions**, Addison-Wesley Professional , Boston, USA, 2003.

[9] Hohpe G: **SOA Patterns – New Insights or Recycled Knowledge**, `http://eaipatterns.com/docs/SoaPatterns.pdf`, Accessed: 22th September 2011.

[10] Li X, Fan Y, Madnick S, Sheng Q Z: **A pattern-based approach to protocol mediation for web services composition**, Information and Software Technology, 2010, 52(3), pp. 304-323.

[11] OASIS: **OASIS Web Services Business Process Execution Language Version 2.0 Primer**, `http://docs.oasis-open.org/wsbpel/2.0/Primer/wsbpel-v2.0-Primer.pdf`, Accessed: 24th October 2011.

[12] OASIS: **OASIS Web Services Business Process Execution Language Version 2.0**, `http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf`, Accessed: 24th October 2011.

[13] Ouyang C, Verbeek E, van der Aalst, W M P, Breutel S, Dumas M, ter Hofstede A H M: **Formal semantics and analysis of control flow in WS-BPEL**, Science of Computer Programming, 2007, 67(2-3), pp. 162-198.

[14] Papazoglou M P, van den Heuvel W: **Service oriented architectures: approaches, technologies and research issues**, The VLDB Journal, 2007, 16 (3), pp. 389-415.

[15] Russel N, van der Aalst W M P, ter Hofstede A H M, Edmond D: **Workflow Resource Patterns: Identification, Representation and Tool Support**, Proceedings of the 17th Conference on Advanced Information Systems Engineering, Portugal, 2005, pp. 216-232.

[16] Russel N, ter Hofstede A H M, Edmond D, van der Aalst W M P. Workflow Data Patterns: **Identification, Representation and Tool Support**, Proceedings of the 24th International Conference on Conceptual Modeling; Berlin, Germany, 2005, pp. 353-368.

[17] Russel N, ter Hofstede A H M, van der Aalst W M P, Mulyar N: **Workflow Control-Flow Patterns: A Revised View**, `http://www.workflowpatterns.com/documentation/documents/BPM-06-22.pdf`, Accessed: 28th January 2011.

[18] Russel N, van der Aalst W M P, ter Hofstede A H M: **Workflow Exception Patterns**, Proceedings of the 18th International Conference on Advanced Information Systems Engineering (CAiSE 06), 2006, pp. 288-302.

[19] Russel N: **Foundations of Process-Aware Information Systems - PhD Thesis**, Brisbane, Queensland University of Technology, 2007.

[20] Schumacher M, Fernandez-Buglioni E, Hybertson D, Buschmann F, Sommerlad P: **Security Patterns – Integrating Security and Systems Engineering**, John Wiley & Sons , Chichester, 2006.

[21] Trowbridge D, Roxburgh U, Hohpe G, Manolescu D, Nadhan, E G: **Integration patterns**, Microsoft Corporation, 2004.

[22] Van Lessen T, Nitzsche J, Leymann F: **Formalising Message Exchange Patterns using BPELlight**, IEEE International Conference on Services Computing (SCC '08), Honolulu, SAD, 2008, pp. 353-360.

[23] Zdun U, Hentrich C, Dustdar S: **Modeling Process-Driven and Service-Oriented Architectures Using Patterns and Pattern Primitives**, ACM Transactions on the Web (TWEB), 2007,1(3), pp. 1-40.

[24] Zdun U: **Pattern-Based Design of a Service-Oriented Middleware for Remote Object Federations**, ACM Transactions on Internet Technology, 2008, 8(3), pp. 1-38.