

Coordinated Web Services Orchestration

Geert Monsieur, Monique Snoeck and Wilfried Lemahieu

Katholieke Universiteit Leuven, Faculty of economics and applied economics

Department of Decision Sciences and Information Management

Leuven Institute for Research on Information Systems (LIRIS)

Naamsestraat 69, 3000 Leuven (Belgium)

Email: {Geert.Monsieur,Monique.Snoeck,Wilfried.Lemahieu}@econ.kuleuven.be

Abstract

In a Business Process Execution Language (BPEL) process definition the sequence of exchanged messages typically originates from the sequence of business process activities and from the need of coordination of those activities across the participants of the process. As such business concerns (e.g. the sequence of business process steps) are often mixed with technical aspects (e.g. the sequence of coordination messages). In this article we present an architecture to separate business and technical concerns, which results in a clearer overview of the high-level business process and improves the flexibility and maintainability of the orchestration architecture. The described architecture depends on existing Web service standards. Different eventing and coordination specifications are discussed. The ultimate architecture is mainly based on the WS-Brokered Notification and WS-Coordination Framework specifications.

1. Structured Web services orchestration

1.1. A spaghetti of messages

Web services orchestration refers to an executable business process that can interact with both internal and external Web services. The interactions occur at the message level. They include business logic and task execution order, and they can span applications and organizations to define a long-lived, transactional, multistep process model [1]. The business process execution language (BPEL) is the leading standard language for orchestration of Web services. A BPEL process defines how multiple service interactions are coordinated to achieve a business goal, as well as the state and the logic necessary for this coordination [2]. Figure 1 shows an example of a business process

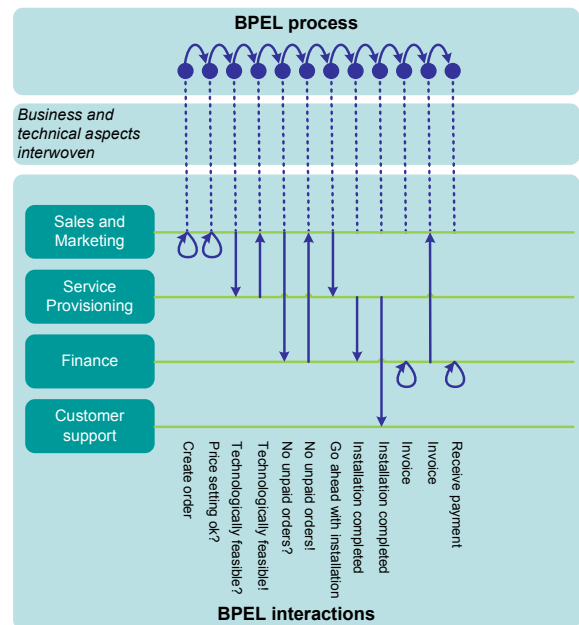


Figure 1. A traditional BPEL process

defined in a BPEL manner. We consider a business process for order handling in a telecommunication company that provides broadband services to its customers. Executing the business process is done by orchestrating four business services (Sales and Marketing, Service Provisioning, Finance and Customer Support). As one can see in figure 1, in a BPEL process definition typically the sequence of business process activities and the sequence of message exchanges needed for the coordination of a specific activity are interwoven. Business concerns are mixed with technical (e.g. coordination) aspects. In this way managing business processes means managing a huge amount of communication messages which is difficult. The necessary

messages for coordinated processing of a business activity included in the BPEL process definition hamper the overview of the high-level business task or process. Defining business processes as sequence constraints on message exchanges is a too low-level task. The challenge is to structure the spaghetti of messages so that business process design and (automated) execution becomes a less painful and complex job.

1.2. Raising the abstraction level

To structure the orchestration of Web services we introduce the concept of business events [3]. From a bottom-up perspective a business event is a grouping of a set of message exchanges. All grouped messages are relevant for one specific activity and match a real-world business phenomenon. We say that the logically related messages constitute a business event. This allows to separate business event sequencing and business event notification. Notification refers to the fact that business events are always related to different parties that need to be notified (the business event participants). These parties are the different business services used for the processing of a business event. However if one looks closely at the message exchanges, one can see that messages are not only exchanged for notification purposes, but often also for validation purposes (e.g. checking if there are unpaid orders). In particular all parties involved in a business event may enforce business rules and constraints as preconditions on the event. If one party finds one or more preconditions to be violated, the entire system rejects the event, and no processing should take place in any of the applications. If processing has already occurred, it is rolled back or compensated, depending on the desired business coordination protocol. In this sense, events incorporate a transactional aspect. And so the required architecture for business event orchestration is different from a fire-and-forget event architecture [4]. In fact the notification aspect of business events differs from the notification of traditional events. Instead of notifying parties of an *occurred* event, notifying business event participants is about notifying there is a *desire* to process a specific business event or activity (thus the business event still needs to occur).

Hence in order to process a business event two aspects need to be taken care of: notification and coordination. In that way from a top down perspective a business event is a business activity which is atomic and which requires a set of (notification and coordination) messages. It is important to understand why business event coordination is necessary. In a business event several aspects are *abstracted* away. One of these

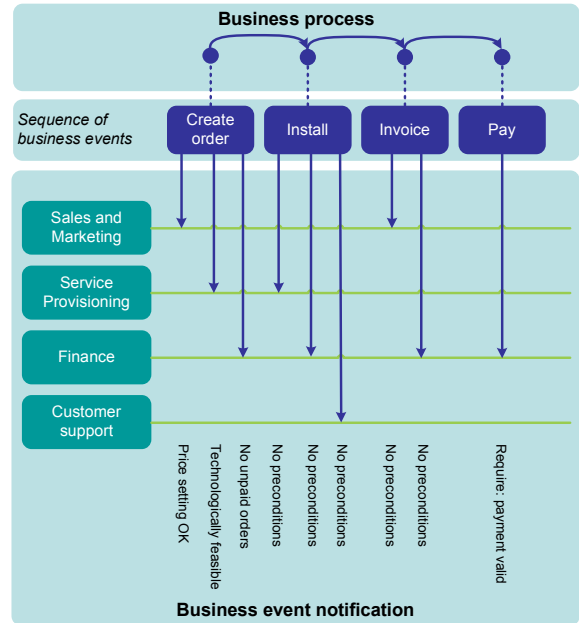


Figure 2. Business event orchestration

abstractions is about the atomicity of business event. When designing at the business level, architects count on the consistent processing of business events. Coordination should be implicitly present. The business demands that events never are processed just partially. In that way business events become high-level units of coordination and atomic units of (inter)action. This allows to use business events as building blocks for business processes at the business process layer (as shown in figure 2 and 3) and the process designer does not need to take care of a number of aspects which are abstracted away.

Introducing the high-level concept of a business event at the business level raises the abstraction level, and reduces complexity. This leads to several benefits for the business. By using *abstract* business events at the business process level it should be easier to reason about the behavior of systems while executing business processes. Similarly to traditional software development we use a number of UML diagrams as a means to abstraction. The goal (of abstraction) is to isolate those aspects that are important for some purpose (e.g. focusing on business processes) and suppress those aspects that are unimportant [5][6]. Important aspects in the context of business events are the global business activity (called a business event) and the participants of a business event. Less important and temporarily suppressed or abstracted aspects are the specific message exchanges needed for the coordinated processing of the

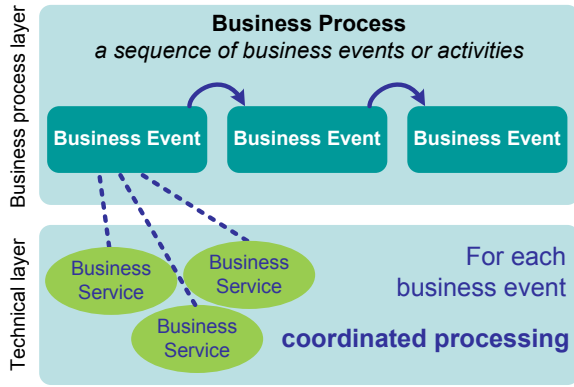


Figure 3. Business layer and technical layer

business activity or event. The abstract business model provides us a *high-level and clear overview* of the business process. Furthermore the use of business events should also make life easier in the context of business process automation and logical consistency checks, horizontal as well as vertical consistency. Horizontal consistency refers to the consistency between different business processes. Vertical consistency is considered as the consistency between a business process and the supporting information system.

Figure 3 summarizes the ideas from above. The business event orchestration architecture consists mainly of two layers: the business process layer and the technical layer. The top layer defines the high-level business process in terms of business events. The bottom layer is responsible for the notification and what we call the **Business Event Coordination (BECO)**¹.

1.3. A model-driven approach

The aspects that have been abstracted away by working with business events still need to be taken care of to realize a complete working system. This boils down to applying the principles of the model driven architecture (MDA). At the conceptual level the business process designer identifies business events and specifies for each business event which components or business services need to be notified to process the business event. This could e.g. be specified in a simple tabular format such as the well-known CRUD tables [7]. In this article we refer to such a table as the *component event table (CET)* [8]. In the context of MDA the combination of a business process (sequence of business events) and a CET can be considered as the *platform independent model (PIM)*. *The platform*

1. <http://beco.econ.kuleuven.be> contains more information about BECO related research

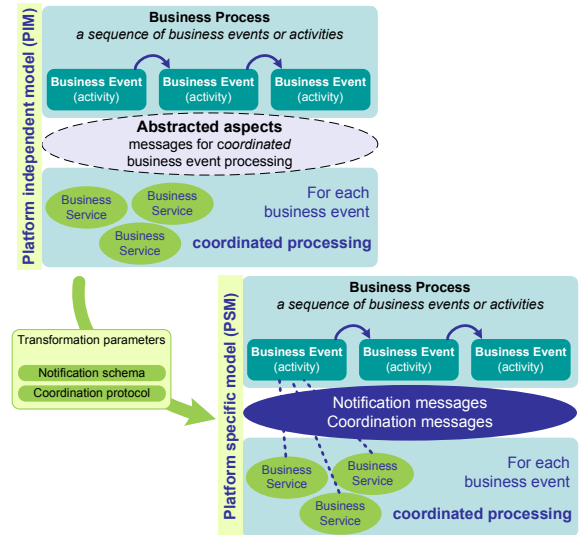


Figure 4. Business event transformation

specific model (PSM) is the complete architecture to orchestrate the business process and includes notification and coordination mechanisms for business events. To transform the high-level PIM into a low-level PSM a clearly defined transformation pattern is required [9]. A typical parameter for the transformation in the context of business events could be e.g. the specific coordination protocol that is applied in the PSM. Figure 4 presents an overview of the model-driven development process for the business event architecture. There are many different specific transformations possible. One can design manually for each business event a specific chain or sequence of messages for the coordinated processing of the event, or one can rely on a central business event dispatching entity. The latter entity applies automatically for all business events a similar predefined transformation pattern to obtain the proper coordination architecture.

The presented layered architecture has the advantage that changing the business process or changing the event processing (e.g. the message notification schema) can occur separately. When there is a change in the high-level business process one can simply apply the transformation pattern to obtain an updated PSM, without having to worry about the notification and coordination aspects of the redesigned business process.

1.4. Structure of the text

In the previous section we have discussed the idea of transforming a high-level business process model (consisting of business events and their participating

services) into a specific technical model with notification and coordination capabilities. In the following sections of this article we will discuss a fixed pattern for realizing the notification and coordination of business events in a Web services context. This pattern could e.g. be used by a central event dispatching entity which takes care of the proper processing of a business event. We present a possible transformation for a Web services platform, focused on an automatic generation of the necessary messages for coordinated processing of the business event. In our solution we try to reuse as much as possible existing Web services standards and specifications.

First we will discuss in section 2 different eventing specifications (WS-Events, WS-Eventing and WS-Notification). Subsequently in section 3 we present a short introduction to existing coordination specifications (WS-Coordination and WS-Coordination Framework). Since all these specifications, when used in isolation, do not meet the requirements of a business event architecture, we present in section 4 how a *combination of an eventing and coordination specification* could be useful to build a business event architecture.

2. Web services and events

2.1. WS-Events

The WS-Events specification consists of 'a set of processing rules for advertising, subscribing, producing and consuming Web services Events' [10]. All authors of this specification are based at Hewlett-Packard. Further development of WS-Events is not on the agenda anymore. Therefore we only give a brief overview of this specification.

WS-Events defines an *event* as a change in the state of a resource or request for processing. The specification describes three main entities. The *event producer* is an entity which generates notifications. These notifications are received by the *event consumers*. The *event broker* is responsible for routing the notifications. Brokers typically aggregate and publish events from several producers. An event broker can also apply some transformation to the notifications it processes. The precise role of the broker remains unclear in WS-events. Subscription requests and notifications are described as (direct) messages between the event producer and the event consumer. The specification does not provide descriptions of entities responsible for subscription or subscription management tasks (such as the subscriber and subscription manager in WS-Notification and WS-Eventing) nor does it distinguish

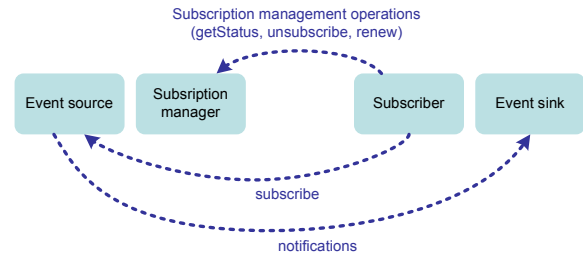


Figure 5. WS-Eventing

between publisher and producer roles (as in WS-Brokered Notification).

2.2. WS-Eventing

The WS-Eventing specification describes a protocol that allows Web services to subscribe to or accept subscriptions for event notification messages [11]. The specification defines four entities in the event notification architecture (see figure 5). An *event source* is considered as a Web service that sends notifications and accepts requests to create subscriptions. A Web service that receives notification messages is referred to as an *event sink*. Before this event sink can receive notifications it's *subscriber* needs to send a subscription creation request to the event source. The subscriber is also responsible for sending subscription management requests (retrieving status information, renewing or deleting subscriptions). These latter requests are sent to and handled by the *subscription manager*.

2.3. WS-Notification

The Web services Notification (WS-Notification) family of specifications defines a framework for event notification in a Web services environment. It consists of three main specifications, namely WS-Base Notification, WS-Brokered Notification and WS-Topics [12].

2.3.1. Terminology and concepts. All WS-Notification specifications use a common terminology for the entities used in the framework definitions. We will give a brief description of the most important concepts.

An event is published by a *publisher*. This entity creates a notification message based on an event it is capable of detecting. The publisher does not have the responsibility for sending the notification message to the appropriate receivers. This task is reserved for the *notification producer*. In some cases, the notification producer also has the role of publisher (thus it creates the notification messages itself). In

the other cases the notification producer is referred to as a *notification broker*. The broker similarly distributes notification messages that were created by a separate publisher. In order to distribute notification messages the notification producer maintains a list of interested and registered *notification consumers*. Expressing interest in some types of notification messages occurs with the use of so called *topics*. For more details about the topic-based subscription mechanism we refer to the WS-Topics specification [12]. A *subscriber* is an entity that sends subscription requests to the notification producer on behalf of a notification consumer. Although it is not necessary, a notification producer can delegate subscription management tasks (retrieving subscription status, unsubscribing and renewing) to a separate entity, a so called *subscription manager*.

2.3.2. WS-Base Notification. Figure 6(a) represents the architecture described in the WS-Base Notification specification. There are three main entities defined, namely a notification producer, a subscriber and a notification consumer. The use of a subscription manager is optional.

As one can see there is no publisher in this setting. The notification producer is responsible for creating *and* sending the notification messages to the consumers. If required the notification producer can delegate subscription management to a subscription manager.

2.3.3. WS-Brokered Notification. Figure 6(b) represents the brokered version of WS-Notification. The central entity in this pattern is the broker. This intermediary decouples publishers from notification producers. It has two main responsibilities, distributing notification messages (the notification producer's task) and managing subscriptions (the subscription manager's task) (see figure 6(b)). Beside these basic functionalities the broker can provide so called additional *added-value functions*. Examples of these functions are logging notification messages, transforming topics or notification message content [13].

2.4. Concluding remarks on eventing

Most eventing standards have a fire-and-forget character. Using a standalone implementation of an eventing specification for our business event architecture is not sufficient. Business events need *coordinated* processing. In order to achieve business event coordination we try to complement an eventing architecture with coordination capabilities.

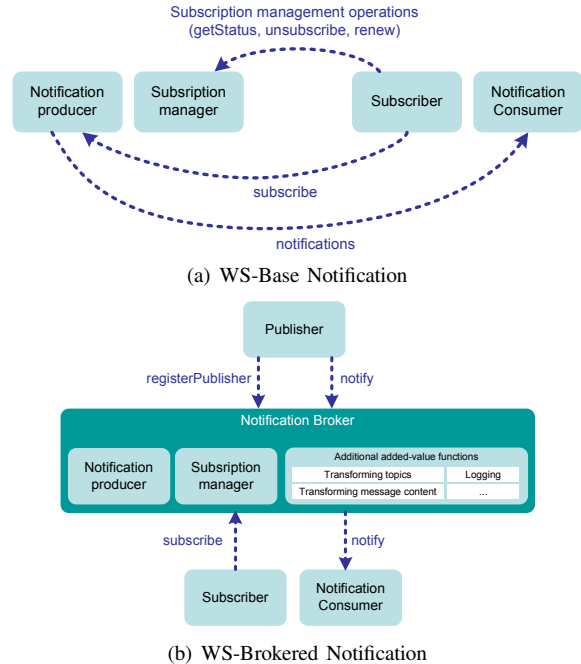


Figure 6. WS-Notification

3. Web services coordination

In this section we will discuss two Web services coordination specifications. Specific transaction or coordination protocols are beyond the scope of this article. The focus is on the generic coordination architecture.

3.1. WS-Coordination (WS-C)

In general, coordination can be seen as the act of one entity (known as the coordinator) disseminating information to a number of participants or components for some domain-specific reason. The reason could be to reach consensus on a decision like in a distributed transaction protocol, or simply to guarantee that all components obtain a specific message, as occurs in a reliable multicast environment [14].

All these kinds of coordination have something in common. The idea is that when components are being coordinated, information known as the *coordination context* is propagated to tie together operations which are logically part of the same coordinated activity. The WS-Coordination (WS-C) specification is built starting from this idea. It defines a *generic* framework which can be used to *propagate context information*, independent of the coordination protocol used.

The main part of the WS-C specification describes the different services and functionalities a coordinator

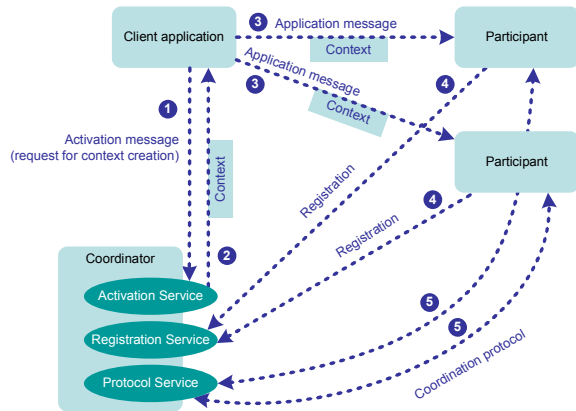


Figure 7. WS-Coordination architecture

should provide. Three services are defined in the specification. The task of the *activation service* is to provide an interface where components can request the creation of a coordination context. Since the propagation of context information - which bundles operations that are part of the same coordinated activity - is necessary before any coordination can occur, the request of context creation can be seen as the activation of the coordination. Once components have received context information they can register for coordination by talking to the *registration service* of the coordinator mentioned in the context information. Actual coordination is realized by protocol communication between the *protocol service* of the coordinator and the participating (registered) components. Context information typically consists of a reference to the registration service of a coordinator, the coordination type and protocol-specific information [15]. Figure 7 represents the overall WS-C architecture. As an illustration we will discuss how one can apply WS-C to implement e.g. the single sign-on protocol. Single sign-on (SSO) is a form of software authentication that enables a user to authenticate once and gain access to multiple components. In terms of the WS-C architecture authentication is done when the client application sends a request for context creation. In case of successful authentication the coordination process is activated. Then the client application receives context information which can be used in communications with all the different components (e.g. printer service or email service). Components use this context information to register with the coordinator. Registration occurs every time a component receives a particular context for the first time. Registration is successful when the coordinator manages to associate a logged on user (and thus an active coordination process) with the details of the registration message (e.g. a ticket code

that was included in the context information). When the user logs out, the coordinator sends via the registration service warnings to the registered components to revoke access rights of the user [14].

3.2. WS-Coordination Framework (WS-CF)

The purpose of the Web services Composite Application Framework (WS-CAF) standardized by OASIS is to define a generic and open framework for applications that contain multiple services used in combination (composite applications). The framework consists of three specifications: WS-Context, WS-Coordination Framework (WS-CF) and WS-Transaction management.

Since we are not interested in the specific coordination protocols (defined in the WS-Transaction management specification) for the moment, but rather in the high-level coordination architecture we only discuss the WS-Context and WS-CF specifications.

3.2.1. WS-Context. The architectures described in WS-C and WS-CF both use the concept of contexts. As discussed earlier a context provides a way to correlate a set of messages into a larger unit of work by sharing common information such as a security token exchanged within a single sign on session. It can be used to identify an activity or a business event. While WS-C defines the management of contexts and the coordination mechanisms together, WS-CAF defines separated specifications for context management and coordination infrastructure. The purpose of WS-Context is to handle and manage context information [16]. Some WS-Context functionalities (in particular the Context Service) match with the concepts defined in the activation service of a coordinator as defined in WS-C. Additionally WS-context defines message exchanges used to query the content of a context or the state of coordination - this happens via the *context manager service*. The latter functionality is especially useful when a context contains a large amount of data and is not supported in WS-Coordination [17].

3.2.2. The coordination framework. Figure 8 presents an overview of the architecture defined in the WS-CF specification. The main difference with WS-C is that WS-CF does not define entities (like the activation service in WS-Coordination) and operations for the creation and management of contexts. Instead it depends on the WS-Context specification for these context related tasks. Roughly speaking WS-C's registration service and protocol service provide the same functionalities as WS-CF's registration service

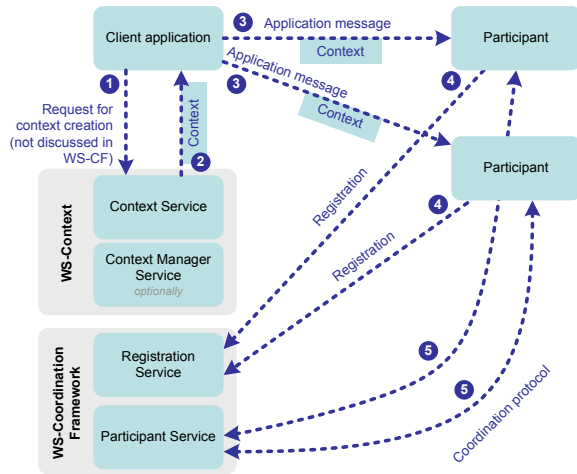


Figure 8. WS-Coordination Framework (WS-CF)

and participant service respectively.

In figure 8 no connection between the WS-Context entities and WS-CF entities (registration and participant service) is shown. In fact the WS-CF specification doesn't discuss how a registration service is located or associated with the Context Service. Although one can surely consider possible coordination protocols which don't require this connection, we believe one can not ignore the possible working relationships between a registration and/or participant service and the context services. In case of the single-sign protocol the registration service should validate received registration messages (in particular the ticket information) with the support of the context services. The context services also need to interact with the registration and/or participant service if the user logs out and all services needs to be informed of this security-related change.

3.3. Concluding remarks on coordination

We conclude that the architectures in WS-Coordination and WS-Coordination Framework (complemented with WS-Context) provide a quite similar solution for the coordination problem, but WS-CF is clearly more generic than WS-C.

4. A Web services based business event architecture

Now it should be clear that we need a combination of an eventing and coordination specification to build a business event architecture.

An event based specification is used for *notifying* business event participants when an entity triggers a

business event and requests processing of a business event. Because of the atomicity property of a business event we also need an additional coordination functionality. The WS-C specification is intended for coordinating activities and it defines an activity as distributed units of work [15]. The WS-CF states that coordination of multiple Web services in choreography may be required to ensure the correct result of a series of operations comprising a single business transaction [18]. Since a business event is precisely an activity that represents a distributed unit of work or a series of operations comprising a single business transaction, it seems rational to use WS-C or WS-CF for business event coordination. The concept of context used in both coordination specifications functions as the glue that binds all low-level one-to-one messages into a high-level business event.

Since WS-Notification is the only eventing standard available with a brokered version - which improves the decoupling of event sources and consumers - we will use WS-Brokered Notification in our business event solution. For the coordination aspect we prefer the more generic WS-Coordination Framework.

4.1. An event-based specification as the basis

WS-Notification defines the publisher as the entity that publishes the event. In the context of business events the publisher acts as the entity that *triggers* (publishes) the business event. This entity can be a business event participant or a higher-level entity like a business process engine (see figure 4). As described in the WS-Notification specification it is the responsibility of the notification producer to send event notification messages to the appropriate event consumers. The notification messages inform the consumers, which are the business event participants, about the triggered business event. The notification producer can retrieve a list of business event participants from the subscription manager. The latter entity stores an overview of business events and the business event participants. In other terms one can state that the subscription manager holds the CET. An entity subscribing at the subscription manager registers itself as a business event participant.

4.2. Adding a coordination mechanism

A fire-and-forget architecture based on an event notification specification (e.g. WS-Notification) is not enough to implement a business event approach. It lacks a coordination mechanism which guards the atomicity of a business event. The basics of WS-C and WS-CF are similar and based on disseminating context

information to accomplish the coordination task. The notification producer should activate a coordination process for each event notification. This activation occurs by sending a request for context creation to the coordination entities (in particular the context service in case of WS-CF). As such the coordination entities can prepare the coordination process. Next, the basic idea is to include the context information in the traditional notification messages sent by the notification producer. In that way event consumers or business event participants receive context information and can register themselves for the coordination of the business event processing (as described in the coordination specifications).

4.3. Complete solution

Figure 9 shows an overview of how one can build a business event architecture with the use of WS-Brokered Notification and WS-Coordination Framework.

First a publisher triggers a business event by sending a message to the notification broker. Then the broker's notification producer activates the coordination (sending a context creation request) and receives context information from the coordination entities. Subsequently the notification producer retrieves the business event participants for the triggered event and can notify the appropriate consumers. Context information is included in the notification messages. As one can see in figure 9 the publisher is also notified of the (business) event it published. As such publishers also participate in the coordination process and are kept informed of the processing status of the business event. Next, the notified business event participants need to register themselves for the coordination process. Overall business event coordination is done by the specified coordination protocol between the coordination entities (connected in the broker) and the business event participants.

5. Conclusions and future work

5.1. Concluding remarks

We have presented a structured and layered architecture to achieve a *coordinated* Web service orchestration. To separate business process concerns of technical issues we introduced the concept of business events at the top layer. A business process is defined by a sequence of business events. The bottom layer is responsible for the (low-level) technical processing of a high-level business event. Hence we have clearly

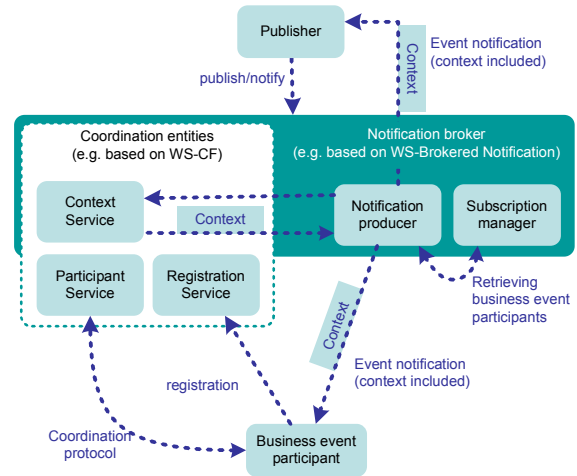


Figure 9. A business event architecture based on WS-Brokered Notification and WS-Coordination Framework

separated business and technical aspects. Business processes (as sequences of business events) as such are only considered at a high level of abstraction, without worrying about low-level coordination concerns. In contrast a typical BPEL business process definition still contains business process aspects at the level of all interaction messages. We believe the use of BPEL is needed at a higher level of abstraction, e.g. for defining the sequences of business events and triggering a specific business event. Low-level notification and coordination occurs *implicitly* by the supported architecture that is described above.

The existing WS-specifications (collectively known as WS-*) are numerous and daunting [19]. All the WS-* specifications fit into someones idea of an overall architecture. The problem is there are different views on this overall WS-architecture. Are all WS-specifications really necessary? How do these standards or specifications fit together? Clearly there is still a lot of work needed to clarify the vague world of WS-* and transform WS-specifications in real *standards*. In this article we tried to contribute in the future convergence of different eventing specifications [20]. Furthermore we aimed to present a better overview of the relationships between business process orchestration and coordination aspects managed by coordination specifications like WS-Coordination and WS-Coordination Framework. Similarly, research of Tai et al. shows how one can create a *composition of coordinated Web services*. Their objective is "to integrate the use of external coordination protocols (specifically, WS-AT and WS-BA) for different activities of a BPEL process

composition, where BPEL defines the scope of the coordination” [21]. Our research is unique in the sense that it clearly separates the high-level business process and the low-level message interactions needed for the processing of the business activities. As a consequence the proposed architecture fits well in the context of the model driven architecture (see section 1.3).

5.2. Future work

Our business event approach has already proven to be valuable on a single nondistributed platform, with the focus on small applications. Transformation of the business events occurred automatically (using fixed transformation patterns) and resulted in a prototype application [8][22].

Furthermore we have tested our business event approach in a real-life application running in a client-server environment [3][23]. In the latter environment the focus was on component coordination. Transforming high-level business events into low-level concepts occurred manually. The solution lacked the support of services and was insufficiently scalable.

In the future we intend to address more types of environments. In particular the use of business events in a Web services context is on our research agenda. The focus will be on the model-driven development of the Web services based business event architecture. With the use of MDA tools (e.g. Compuware’s OptimalJ or IBM’s Software Architect) we hope to define useful transformation patterns for the architecture discussed in this article. In that way we hope to obtain some kind of formal model of our business event approach in a Web services context. Subsequently we plan to build a prototype of the Web services based business event architecture.

Although this article presents a solution to integrate coordination and event-based business process execution, a real business event orchestration also requires a *specific* coordination protocol. Similarly, WS-Coordination and WS-Coordination Framework are generic models for coordination and several specific coordination protocols can be plugged in (e.g. atomic transaction and business activity) [24]. The relationship between the concept of business events and a specific coordination protocol is a topic for further research.

Acknowledgements

This research was part of a project funded by the Research Fund K.U.Leuven (OT 05/07), whose support is gratefully acknowledged.

References

- [1] C. Peltz, “Web services orchestration and choreography,” *Computer*, vol. 36, no. 10, pp. 46–52, 2003.
- [2] Oasis, “Web services business process execution language 2.0 (bpel),” 2006.
- [3] W. Lemahieu, M. Snoeck, F. Goethals, M. D. Backer, R. Haesen, J. Vandenbulcke, and G. Dedene, “Coordinating cots applications via a business event layer,” *IEEE Softw.*, vol. 22, no. 4, pp. 28–35, 2005.
- [4] P. Eugster, P. Felber, R. Guerraou, and A. Kermarrec, “The Many Faces of Publish/Subscribe,” *ACM Computing Surveys*, vol. 35, no. 2, pp. 114–131, 2003.
- [5] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen, *Object-oriented modeling and design*. NJ, USA: Prentice-Hall, Inc., 1991.
- [6] E. V. Berard, “Abstraction, encapsulation, and information hiding,” *E Berard Essays on Object-Oriented Software Engineering*, vol. 1, 1993.
- [7] J. Martin, *Information Engineering: Introduction*. Prentice Hall, NJ, USA, 1989.
- [8] M. Snoeck, *Object-Oriented Enterprise Modelling with Merode*. Leuven University Press, 1999.
- [9] A. G. Kleppe, J. Warmer, and W. Bast, *MDA Explained: The Model Driven Architecture: Practice and Promise*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2003.
- [10] Hewlett-Packard, “Web services events 2.0,” 2003.
- [11] BEA Systems, Computer Associates, IBM, Microsoft, TIBCO Software, “Web services eventing,” 2004.
- [12] Akamai Technologies, Hewlett-Packard, IBM, Sonic Software, SAP, Tibco Software, “Web services notification 1.0),” 2004.
- [13] P. Niblett and S. Graham, “Events and service-oriented architecture: the oasis web services notification specifications,” *IBM Syst. J.*, vol. 44, no. 4, pp. 869–886, 2005.
- [14] M. Little and J. Webber, “Introducing ws-coordination,” *Web Services Journal*, 2003.
- [15] Arjuna, BEA Systems, Hitachi, IBM, IONA, Microsoft, “Web services coordination version 1.0,” 2005.
- [16] Oasis, “Web services context,” 2006.
- [17] F. Leymann and S. Pottinger, “Rethinking the coordination models of ws-coordination and ws-cf,” in *Proc. of the Third European Conf. on Web Services*. Washington, USA: IEEE Comp. Soc., 2005, p. 160.
- [18] Oasis, “Web services coordination framework,” 2006.
- [19] S. Vinoski, “WS-Nonexistent Standards,” *IEEE Internet Computing*, vol. 8, no. 6, pp. 94–96, 2004.
- [20] K. Cline, J. Cohen, *et al.*, “Toward Converging Web Service Standards for Resources, Events, and Management,” 2006.
- [21] S. Tai, R. Khalaf, and T. Mikalsen, “Composition of coordinated web services,” *Proc. of the 5th ACM/IFIP/USENIX international conf. on Middleware*, pp. 294–310, 2004.
- [22] G. Monsieur, M. Snoeck, R. Haesen, and W. Lemahieu, “PIM to PSM transformations for an event driven architecture in an educational tool,” in *Proc. of European Workshop on Milestones, Models and Mappings for Model-Driven Architecture (3M4MDA)*, 2006.
- [23] W. Lemahieu, C. Michiels, and M. Snoeck, *Integration of third-Party applications and Web-Clients by Means of an enterprise Layer*. Idea Group, 2003.
- [24] M. P. Papazoglou, “Web services and business transactions,” *World Wide Web*, vol. 6, no. 1, pp. 49–91, 2003.