Structured Services Composition: design and implementation

Geert Monsieur, Monique Snoeck, and Wilfried Lemahieu

Katholieke Universiteit Leuven
Faculty of Business and Economics
The Leuven Institute for Research on Information Systems (LIRIS)
Naamsestraat 69, 3000 Leuven (Belgium)
`geert.monsieur,monique.snoeck,wilfried.lemahieu@econ.kuleuven.be`

Abstract. Modern services composition languages such as e.g. the Business Process Execution Language (BPEL) define business processes as sequence constrains on message exchanges. These types of process descriptions are often very complex, because they incorporate many low-level, technical details. As a result, the high-level overview of the business process is easily lost. Therefore we provide a way to structure the composition and orchestration of business services. The structuring technique is based on the concept of business events which is the result of previous research. In this paper we propose the design and implementation of a specific business event architecture that makes structured services composition possible. The implementation in a Web services environment functions as a proof-of-concept of our approach.

Keywords: Services Composition, Orchestration, Business Processes

1 Introduction and problem situation

1.1 Information systems supporting business processes: a need for structured composition

Essentially, business processes define how tasks are executed in order to reach some business goal. The idea of business process modelling and the implementation of business processes as a separate layer on top of information systems, dramatically changed the architectural principles of software development. Until recently, information system support for the tasks in a business process was often realised by means of large monolithical applications that include the rules governing the execution of tasks into the programming code. This entails very unflexible systems that require a lot of time to be adapted to the ever changing business. Component based development was a first step towards a more flexible information system architecture, but the idea of reconfigurable systems really gained ground with the concept of service oriented architectures.
The basic idea behind the service oriented architecture is that business processes should be supported by highly independent, composable and reconfigurable services. This idea changed the mission of software engineers: software engineers

are expected to understand the business so they can devise appropriate business services and mechanisms to aggregate these services. Obtaining this goal would dramatically increase flexibility, since this way every (changing) business scenario would easily be supported by (re)assembling the appropriate business services. For business people it is important that the services used to support their business processes offer functionality that directly contributes to the business goals.

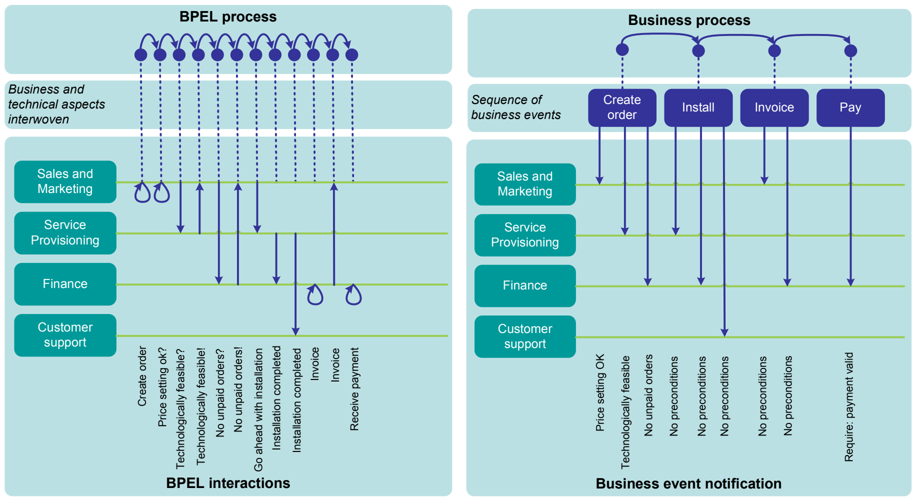
Beautiful as this concept can be, it is far from easy to realise. Processes are hierarchical in nature: high level processes are iteratively refined to lower level processes down to the very details of collaboration protocols that define who sends information to whom and when. Finally, these lower level processes are used as the basis for business process enactment, e.g. by being translated to BPEL. As a result, the implemented processes only contain the most detailed description of a business process, resulting in the loss of the higher level view. It is therefore important that process implementation documents can be traced back to the higher level business processes they originate from. This traceability can be improved by following a structured approach to business process implementation and adopting the right level of granularity for the transition from process to service.

As an example, consider a simple process where at some point a document needs to be signed by three parties. The higher level process will contain a task *sign*, whereas the lower level implementation will split this into three tasks, *sign_by_a*, *sign_by_b* and *sign_by_c*, and specify the details, of who triggers the signature task, who signs first, who sends the document to whom, ... and so on. Especially when this sequence is part of a larger scale process, one easily gets lost in the details of messages sent back and forth between parties.

In the next section, we very briefly illustrate the structuring concept that we propose by means of real-life case. The structuring concept entails aspects both for the modelling of processes as for the implementation of service composition. Full details of the modelling approach can be found in [7, 16–18]. In this paper we present the high level implementation architecture and a proof of concept by means of web services. Furthermore we demonstrate that by using a mechanism of dynamic subscription, services can easily be plugged in and out the service composition.

1.2 Modern way of composition: Web services orchestration

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, multi-step process model [14]. 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 [11]. Figure 1(a) shows an example of a business process defined in a typical BPEL manner.



(a) A traditional BPEL process (b) Business event orchestration

Fig. 1. Unstructured versus structured orchestration

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(a), 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 fact that messages required for coordinated processing of a business activity are included in the BPEL process definition hampers 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.

2 Towards a structured services composition: raising the abstraction level

To structure the composition of services we introduce the concept of business events [7]. 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. 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). As a result, 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. In this way the required architecture for business event orchestration is different from a fire-and-forget event architecture [3].

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 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 (as shown in figure 1(b)) 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. 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 irrelevant [2, 15]. Important aspects in the context of business events are the global business activity (called a business event) and the participants of a business event. This can easily be respresented by a so called *service event table*. In this table the rows are about business events and the columns refer to business services. If a business service is a business event participant the corresponding cell is marked. Temporarily suppressed or abstracted aspects are the specific messages exchanges needed for event notification and the coordinated processing of the business activity or event.

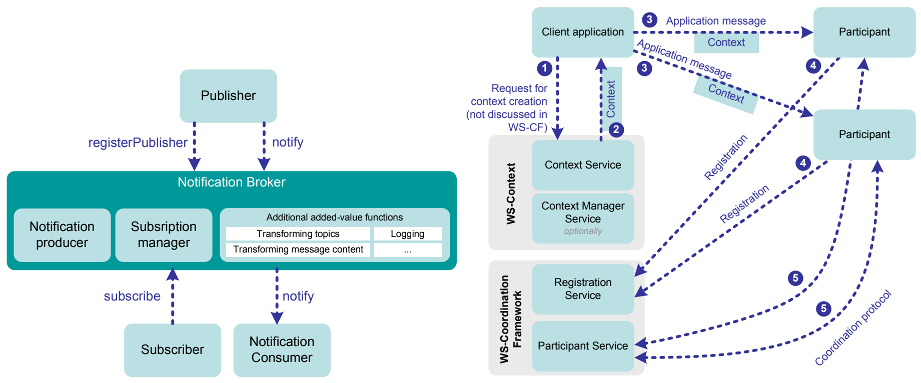
Previous research has demonstrated that the concept of business event based coordination can be mapped to different types of platforms [7]. Furthermore the use of business events also make life easier in the context of business process verification, both in terms of horizontal consistency as in terms of vertical consistency [1, 17]. Horizontal consistency refers to the mutual compatibility of different business processes (e.g. of collaborating partners). Vertical consistency is considered as checking the compatibility between a business process and the supporting information system.
In previous research we used a descriptive evaluation method [5] to validate the design of the event based coordination architecture in a (Web) services environment [4]. In this research we further validate the concept by implementing a proof of concept for a Web services based business event architecture. We believe Web services are very useful to apply the ideas of (structured) services composition. In the next section we provide a high-level overview of our business event architecture. Subsequently, in section 4, we focus on the details of the implementation and illustrate it with our running example. Finally, in section 5 we draw some conclusions.

3 High-level design of a business event architecture

As noted before our business event architecture is based on a combination of two existing Web services standards. The choice of this combination of Web services specifications is the result of previous research [4]. In the following subsection we briefly summarize the aspects of WS-Brokered Notification (Figure 2(a)) and WS-Coordination Framework (Figure 2(b)). As we will illustrate in section 3.2 these specifications play a central role in our design of a business event architecture.

3.1 Reuse of existing Web services standards

WS-Brokered Notification WS-Brokered Notification is part of the WS-notification family of specifications that defines a framework for event notification in a Web services context. Figure 2(a) gives an overview of the entities defined in WS-Brokered Notification. We briefly discuss the different entities.
An event is published by a so called *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*. This entity distributes notification messages that were created by the publisher. In order to distribute notification messages the notification producer should make use of a list of interested and registered *notification consumers*. This list is kept by the so called *subscription manager*. The latter entity is responsible for subscription management tasks (retrieving subscription status, unsubscribing and renewing). A *subscriber* is an entity that sends subscription requests to the notification producer on behalf of a notification consumer. To structure the overal



(a) WS-Brokered Notification (b) WS-Coordination Framework

Fig. 2. Reused Web services specifications

eventing architecture one can use the concept of a *notification broker* which bundles the notification producer and subscription manager. In summary, the broker has two main responsibilities, distributing notification messages (the notification producer's task) and managing subscriptions (the subscription manager's task) (see figure 2(a)). Beside these basic functionalities the broker can provide additional *added-value functions*. Examples of these functions are logging notification messages, transforming topics or notification message content [10].

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 defined in WS-CAF we only discuss the WS-Context and WS-CF specifications. Figure 2(b) presents an overview of the architecture defined in the WS-CAF specification.
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 [9]. 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 Framework 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. As such 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. The purpose of WS-Context is to handle and manage context information [12]. It provides an interface (context service) where components can request the creation of a coordination context. Since the propagation of context information is necessary before any coordination can occur, the request of context creation can be seen as the activation of the coordination. 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 [8]. Once components have received context information they can register for coordination by talking to the *registration service* mentioned in the context information. Actual coordination is realized by protocol communication between the *participant service* and the participating (registered) services. Context information typically consists of a reference to the registration service of a coordinator, the coordination type and protocol-specific information [13].

3.2 A Web services based business event architecture

This subsection shows how one can construct a Web services based business event architecture using an eventing specification (WS-Brokered Notification) and a coordination specification (WS-Coordination Framework).
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-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 [13]. Since a business event is precisely an activity that represents a series of operations comprising a single business transaction, it seems rational to use WS-CF for business event coordination. The concept of context functions as the glue that binds all low-level one-to-one messages into a high-level business event.

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. 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 service event table. An entity subscribing at the subscription manager registers itself as a business event participant.

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-CF are 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). 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). In the rest of this paper when we refer to the business event context we refer to the context concept defined in the WS-CF specification.

Complete design Figure 3 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 by sending a context creation request and receives context information from the context service. 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. Next, the notified business event participants need to be registered for the coordination process. If we would apply the actions as defined WS-Coordination framework strictly, the business event participants should register themselves with the registration service. However, when using a business event approach we want to be sure that *every* business event participant (as marked in the service event table) is involved in the event coordination process. This implies letting business event participant register themselves is probably not the best way of working, since it does not guarantee that *every* business event participant will be registered and involved in the processing of a business event. Therefore, it seems more suitable to delegate the registration process to the notification producer because this entity is already informed of which business services are business event participants (this was needed for notifying the correct business services). Overall business event coordination is done by the specified coordination proto-

col between the coordination entities (connected in the broker) and the business event participants.

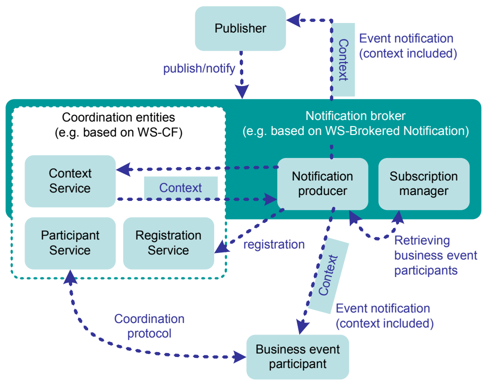

Publisher
publish/notify
Context
Event notification (context included)
Coordination entities (e.g. based on WS-CF)
Notification broker (e.g. based on WS-Brokered Notification)
Context Service
Context
Notification producer
Subscription manager
Participant Service
Registration Service
registration
Retrieving business event participants
Context
Event notification (context included)
Coordination protocol
Business event participant

Fig. 3. A business event architecture based on WS-Brokered Notification and WS-Coordination Framework

4 Implementation of the business event architecture

Although the previous section gives already a good idea how one can construct a business event architecture we believe it is also valuable to add a proof-of-concept to our presented business events approach. Therefore we have implemented the described business event architecture and tested it with a real-life case.

4.1 Java Web services and graphical user interfaces

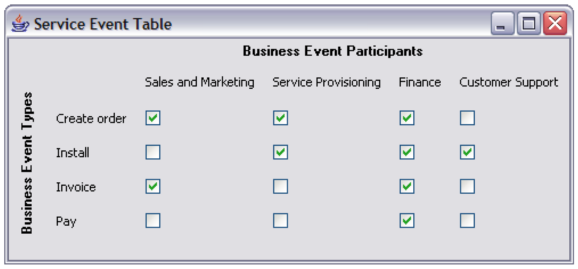
Java Web services The implementation of the business event architecture and example business services (the business event participants) is done in Java. All Web services are implemented as Java Session Beans which were deployed on a JBoss Application Server 4.2.2. Behind the scenes, all persistent information relating to business event types, business event instances, business event participants, business event contexts and registrations is handled by entity beans. The entities were mapped on a Postgresql 8.2 databases using the Hibernate persistence framework. Each service used different databases to increase the loose coupling between the services. To simplify the test environment all Web services were running on the local application server, but since the loose-coupling nature of (Web) services it is quite easy to run the business services on different locations and machines and still achieving *coordinated* and *structured* business services composition.
In the implementation we ignored the Publisher role as mentioned in the WS-Brokered notification specification. The focus in our implementation was on the

coordinated processing of business events. Therefore in the test implementation it is only possible to trigger business event types starting in the notification producer. There is no so called publisher in our implementation setting. A well-equipped business processes supporting information system would have an additional service running to coordinate the sequencing of different business events. In that way we could really test a full business process, but we believe the most important task is to process the business events separately in a coordinated way. If that is proven to work correctly, business events can be used as building blocks for business processes. Then it is only a small step to implement a system that triggers business events in a row as defined in the business process.

Graphical user interfaces To evaluate the implementation we made use of a few graphical user interfaces. We created graphical user interfaces for the notification producer, the subscription manager and the different business services or business event participants. Using the notification producer's interface it is possible to trigger business events and see the current status of processing of business event instances. One can cleary see if a business event instance is still in coordination or is already (un)succesfully finished. Via the subscription manager you can add business event types (giving a name), business event participants (giving a name and a WSDL location) and manage the services event table. The user interface of a business service or business event participant allows to see received business events context or business event notifications and observe the current status of processing of business event instances. Furthermore it provides the test user a way of manually indicating whether preconditions considering a business event instance are violated or not. For more details about the interfaces we refer to the following section that discusses a real-life business example.

4.2 Telecommunication company case as test example

In the beginning of this article we mentioned a business process for order handling in a telecommunication company that provides broadband services to its customers (see figure 1(b)). The business process is composed of four business services (Sales and Marketing, Service Provisioning, Finance and Customer Support). As one can see in figure 1(b) the business process can be described with a specific sequence of business events: *create order*, *install*, *invoice* and *pay*. It is important to notice the different preconditions that comes with e.g. create order event: Price setting ok? Technologically feasible? No unpaid orders? Each of these preconditions are related to one specific business service. These business services are called the business event participants. This means these business services need to be notified of the desire to process the business event and the business event requires coordinated processing in these services. As discussed earlier the connections between business events and business services or business event participants are summarized in the service event table. This table is part of the subscription manager. The user interface to manage the service event table is shown in figure 4.



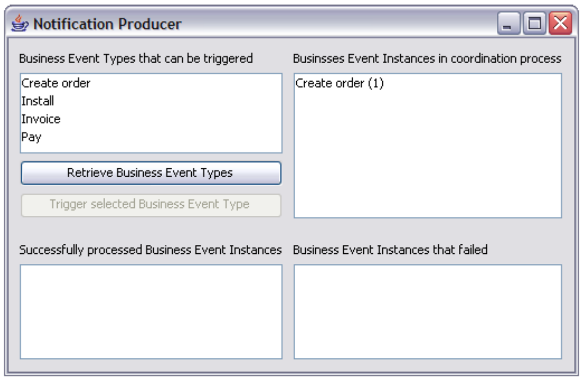
Service Event Table

Business Event Types	Sales and Marketing	Service Provisioning	Finance	Customer Support
Create order	✓	✓	✓	☐
Install	☐	✓	✓	✓
Invoice	✓	☐	✓	☐
Pay	☐	☐	✓	☐

Fig. 4. The services event table: which business services are involved in processing a business event

4.3 Details of processing a business event instance

In this subsection we will discuss the details of processing a business event instance. In particular we will focus on the processing of the *create order* and *pay* business events of the example described in section 4.2. We will explain the processing by referring to the overal architecture (see figure 3) and some screenshots of our test implementation (see figures 5, 6(a), 6(b), 6(c), 6(d) and 7).
Firstly we will examine the processing of the *create order* business event. In



Notification Producer

Business Event Types that can be triggered

Create order
Install
Invoice
Pay

Retrieve Business Event Types
Trigger selected Business Event Type

Businsses Event Instances in coordination process

Create order (1)

Successfully processed Business Event Instances

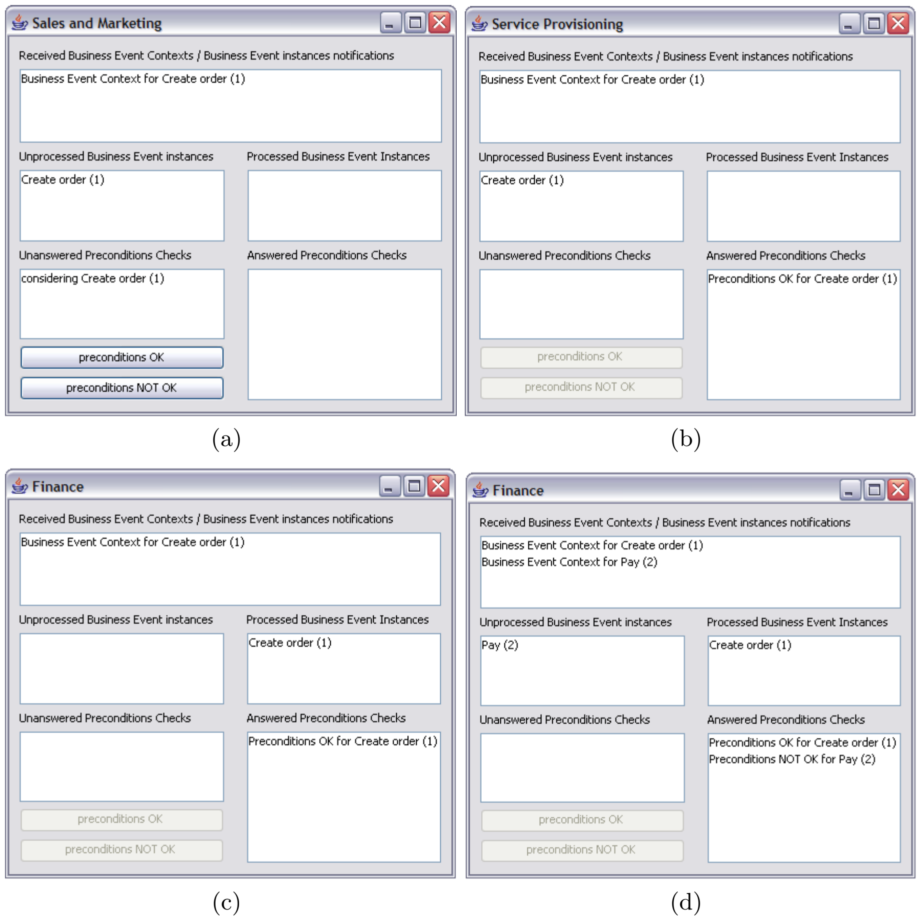
Business Event Instances that failed

Fig. 5. The notification producer after triggering Create order event

figure 5 one can see that the user interface of the notification producer indicates that the test user has triggered a business event type, namely the *create order* event. This means there is a business event instance (here with id 1) that is being coordinated. More precisely a lot of activities are happening after someone triggered a business event. Firstly the notification producer activates the coordination process by sending a request for business event context creation to the context service (see figure 3). Once the coordination context is at hand the notification producer can start sending out notifications to the appropriate event consumers or business event participants. To know which business services are business event participants for the *create order* event the notification producer can call the subscription manager. The latter service indeed holds the service event table (see figure 4). One can observe that it is quite easy to dynamically add a business service as a business event participant by using the check boxes shown in figure 4. We also developed a graphical interface for our subscription manager service which allows adding business event types and business services at runtime. Adding a business event type only requires a name for the business event type, while adding a business service requires beside a name also a WSDL location. All entered business event types and business services are automatically included in the services event table. In the services event table window one can subscribe business services for business event types by marking a specific cell in the table (see check boxes in figure 4).

If one looks closely at the *create order* row in the service event table one should notice that there are three business event participants for this business event. So these three business services (Sales and Marketing, Service Provisioning and Finance) receive the business event notifications which include also the business event context. In figure 6(a), 6(b) and 6(c) one can see these service did receive the notification and context. Next the notification producer registers the business event participants with the registration service so as to be included in the coordination process (see figure 3)). Once all business event participants are registered, the notification producer gives the participant service the green light to start coordinating the processing of the business event (see figure 3). In the test environment we only supported a simple twho-phase commit coordination protocol. This means the participant service starts with asking every business event participant to check preconditions for the *create order* event. As such you can see e.g. in figure 6(a) that the Sales and Marketing service (one of the three business event participants) has one unanswered preconditions check considering business event instance *create order (1)*. Next it is up to the test user to make a decision to send an *ok* message or a *not ok* message back to the participant service. For example in figure 6(b) one can see that the test user (for the Service Provisioning service, yet another business event participant for *create order*) did answer with an *ok* message considering the business event instance *create order (1)*. The same thing is shown in figure 6(c) regarding the Finance service. There one can also notice that an *ok* message is sent back to the participant service, but one could also notice the business event instance is totally processed in the meantime (see the processed business event instances box in figure 6(c)). This means each business event participant has answered the preconditions check with an *ok* message and the participant service had commanded the three business services to complete the processing of the business event instance. The fact that the business event instance *create order (1)* is successfully processed can also be seen in the notification producer's interface (see figure 7).

Next we also give an example where not all businss event participant sent an *ok* message back to the participant service when the latter service is coordinating a business event instance. In particular one can observe the processing of a business event instance *pay (2)* in figure 6(d). In this case the test user for the Finance service did indicate the service does not agree with the preconditions. This means the business event instance *pay (2)* cannot be processed (as one can see in figure 7).



(a) Sales and Marketing

Received Business Event Contexts / Business Event instances notifications
Business Event Context for Create order (1)

Unprocessed Business Event instances
Create order (1)

Processed Business Event Instances

Unanswered Preconditions Checks
considering Create order (1)

Answered Preconditions Checks

(b) Service Provisioning

Received Business Event Contexts / Business Event instances notifications
Business Event Context for Create order (1)

Unprocessed Business Event instances
Create order (1)

Processed Business Event Instances

Unanswered Preconditions Checks

Answered Preconditions Checks
Preconditions OK for Create order (1)

(c) Finance

Received Business Event Contexts / Business Event instances notifications
Business Event Context for Create order (1)

Unprocessed Business Event instances

Processed Business Event Instances
Create order (1)

Unanswered Preconditions Checks

Answered Preconditions Checks
Preconditions OK for Create order (1)

(d) Finance

Received Business Event Contexts / Business Event instances notifications
Business Event Context for Create order (1)
Business Event Context for Pay (2)

Unprocessed Business Event instances
Pay (2)

Processed Business Event Instances
Create order (1)

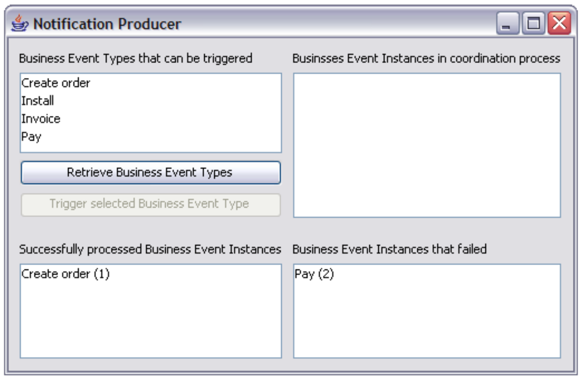
Unanswered Preconditions Checks

Answered Preconditions Checks
Preconditions OK for Create order (1)
Preconditions NOT OK for Pay (2)

Fig. 6. Business Services (Business Event Participants)

5 Conclusions

As mentioned in the introduction (see subsections 1.1 and 1.2) an implemented business process only contains the most detailed description of a business process, resulting in the loss of the higher level overview. We believe defining business processes purely as sequence constraints on message exchanges (e.g. in a typical


Notification Producer
Business Event Types that can be triggered
Retrieve Business Event Types
Trigger selected Business Event Type
Businsses Event Instances in coordination process
Successfully processed Business Event Instances
Business Event Instances that failed

Create order
Install
Invoice
Pay

Create order (1)

Pay (2)

Fig. 7. The notification producer - *create order* succeeded while *pay* failed

BPEL manner) is a too low-level description. Furthermore these descriptions are often too complex to understand for business users and adapting the implemented business processes to changes imposed by business analysts can be a very time consuming and challenging task. Therefore we provided an approach to structure the way of composing and orchestrating business services. The presented approach is based on so called business events which group messages that are relevant for one specific business activity and match a real-world business phenomenon. For each business event we require a coordinated processing so that business users can use business events as atomic building blocks to make up a business process. Business events are already proven to be valuable in a nondistributed environment for the development of small applications as well as component and service-oriented architectures [6, 7, 16–18]. In previous research we discussed the idea of using a combination of a notification and coordination specification to implement a Web services based business event architecture [4]. In this article we presented a proof-of-concept of this business event architecture. We believe our Web services based business event architecture is valuable because we have successfully tested it using a real-life business process. Furthermore by using the developed graphical interfaces it is quite easy to test and discover the strengths of our implementation. For example it is very interesting to see that business services can be added dynamically as business event participants. In the future we plan to further extend the business event architecture e.g. with support for so called business *services types*. Similar to the difference between business event instances and business event types we believe it can be useful to introduce the concept of business *services types*. In that way it is possible to model business cases where e.g. an online book store does not want to use a specific shipper service for all orders, but wants to select a specific shipper *instance* at run-time (based on some parameters e.g. the lowest price).

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. M. De Backer. *The use of petri net theory for business process verification*. PhD thesis, PhD dissertation, Faculty of Business and Economics, K.U.Leuven, 2007.
2. E.V. Berard. Abstraction, encapsulation, and information hiding. *E. Berard Essays on Object-Oriented Software Engineering*, 1, 1993.
3. P.T.H. Eugster, P.A. Felber, R. Guerraouo, & A.M. Kermarrec. The Many Faces of Publish/Subscribe. *ACM Computing Surveys*, 35(2):114–131, 2003.
4. M. Snoeck, G. Monsieur & W. Lemahieu. Coordinated web services orchestration. *Proceedings of the IEEE International Conference on Web Services*, 775–783, 2007.
5. A.R. Hevner, S.T. March, J. Park, & S. Ram. Design Science in Information Systems Research. *MIS Quarterly*, 28(1):75–105, 2004.
6. W. Lemahieu, M. Snoeck, C. Michiels, & F. Goethals. An Event Based Approach to Web Service Design and Interaction. *5th Asia-Pacific Web Conference, Apweb 2003, Xian, China, April 23-25, 2002, Proceedings*, 2003.
7. W. Lemahieu, M. Snoeck, F. Goethals, M. De Backer, R. Haesen, J. Vandenbulcke, & G. Dedene. Coordinating COTS applications via a business event layer. *IEEE Softw.*, 22(4):28–35, 2005.
8. F. Leymann & S. Pottinger. Rethinking the coordination models of WS-Coordination and WS-CF. *Proceedings of the Third European Conference on Web Services*, page 160, Washington, DC, USA, 2005. IEEE Computer Society.
9. M. Little & J. Webber. Introducing WS-Coordination. *Web Services Journal*, May 2003.
10. P. Niblett & S. Graham. Events and service-oriented architecture: the oasis web services notification specifications. *IBM Syst. J.*, 44(4):869–886, 2005.
11. Oasis. Web services Business Process Execution Language version 2.0 (BPEL), May 2006.
12. Oasis. Web services Context (WS-Context), August 2006.
13. Oasis. Web services Coordination Framework (WS-CF), August 2006.
14. C. Peltz. Web services orchestration and choreography. *Computer*, 36(10):46–52, 2003.
15. J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, & W. Lorensen. *Object-oriented modeling and design*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1991.
16. M. Snoeck. *Object-Oriented Enterprise Modelling with Merode*. Leuven University Press, 1999.
17. M. Snoeck, W. Lemahieu, F. Goethals, G. Dedene, & J. Vandenbulcke. Events as Atomic Contracts for Application Integration. *Data and Knowledge Engineering*, 51(1):81–107, 2004.
18. M. Snoeck. *On a Process Algebra Approach to the Construction and Analysis of MERODE-Based Conceptual Models*. PhD thesis, PhD dissertation, Faculty of Science, Dept. of Computer Science, K.U.Leuven, 1995.