KATHOLIEKE UNIVERSITEIT
LEUVEN

Faculty of Business and Economics

Web services specifications on eventing and coordination

Geert Monsieur, Monique Snoeck and Wilfried Lemahieu

DEPARTMENT OF DECISION SCIENCES AND INFORMATION MANAGEMENT (KBI)

KBI 1026

Web services specifications on eventing and coordination

Geert Monsieur, Monique Snoeck, Wilfried Lemahieu
Faculty of Business and Economics
Katholieke Universiteit Leuven
firstname.lastname@econ.kuleuven.be

January 15, 2007

1 Existing event-based Web services standards

1.1 WS-Events

The WS-Events specification consists of 'a set of processing rules for advertising, subscribing, producing and consuming Web services Events' [4]. 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 between publisher and producer roles (as in WS-Brokered Notification).

1.2 WS-Eventing

The WS-Eventing specification describes a protocol that allows Web services to subscribe to or accept subscriptions for event notification messages [3]. The specification defines four entities in the event notification architecture

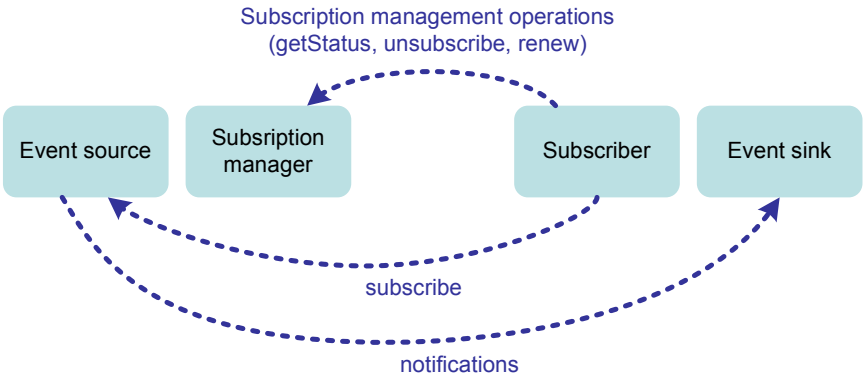


Figure 1: WS-Eventing

(see figure 1). 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*.

1.3 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 [1].

1.3.1 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.

WS-Notification does not specify what a *event* is or is not, nor does it define the concrete relationship between an event and the notification messages that are used to describe the event. In any case an event is some occurrence within a (web) service or component of interest to third parties - most of the time other services or components. 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 pro-

ducer also has the role of publisher (thus it creates the notification messages itself). In the other cases the notitication 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 [1]. 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*.

1.3.2 WS-Base Notification

Figure 2(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.

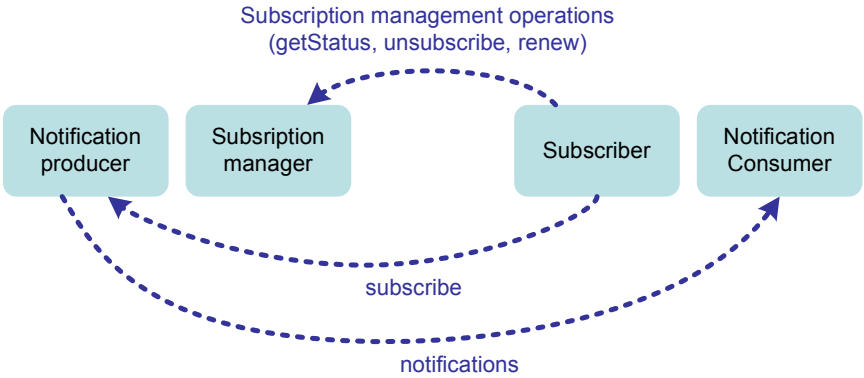
As one can see there is no publisher in this setting. The notification producer is responsible for creating *and* sending the notification messages to the consumers. If required the notification producer can delegate subscription management to a subscription manager.

1.3.3 WS-Brokered Notification

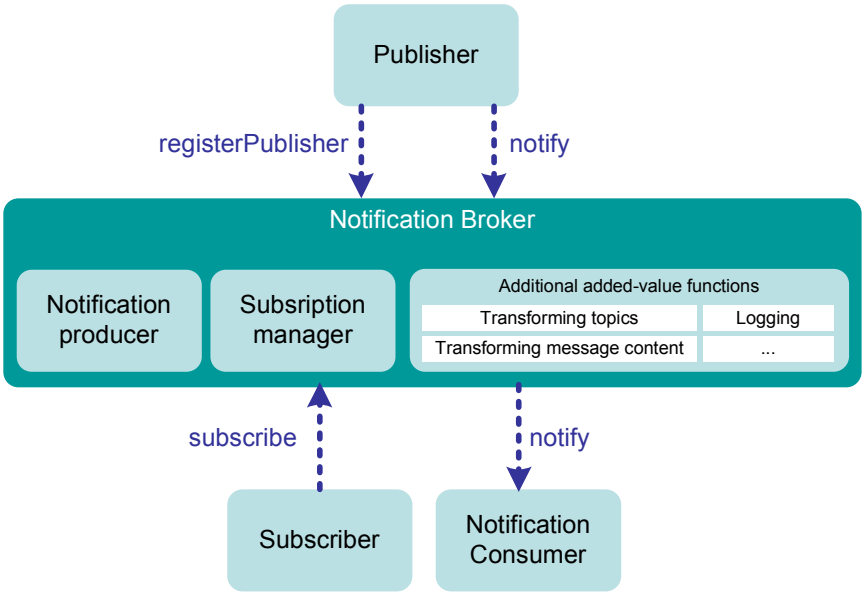
Figure 2(b) represents the brokered version of WS-Notification. The central entity in this pattern is the broker. This intermediary decouples publishers from notification producers. It has two main responsibilities, distributing notification messages (the notification producer's task) and managing subscriptions (the subscription manager's task) (see figure 2(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 [7].

2 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.



(a) WS-Base Notification



(b) WS-Brokered Notification

Figure 2: WS-Notification

2.1 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 [6].

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 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 [2]. Figure 3 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 reg-

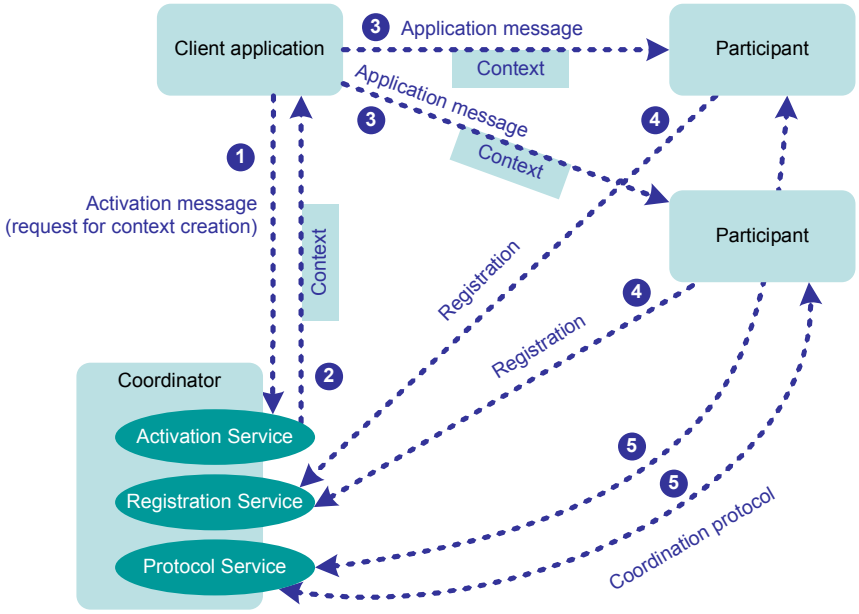


Figure 3: WS-Coordination architecture

istration service warnings to the registered components to revoke access rights of the user [6].

The specification also allows that applications or components use their own coordinators to communicate coordination aspects. In this case the components forward context information to their coordinator. Subsequently the component's coordinator registers with the coordinator mentioned in the context details. For more details about coordination with multiple coordinators we refer to the standard specification [2].

2.2 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.

2.2.1 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 [8]. 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 [5].

2.2.2 The coordination framework

Figure 4 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 and participant service respectively.

In figure 4 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.

References

[1] Akamai Technologies, Hewlett-Packard, IBM, Sonic Software, SAP,

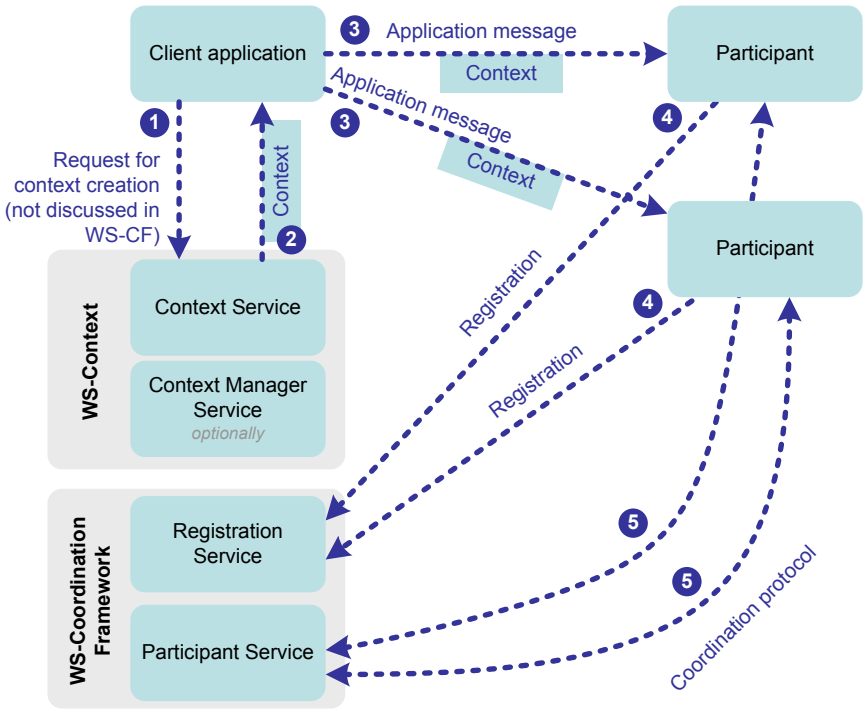


Figure 4: WS-Coordination Framework (WS-CF)

Tibco Software. Web services notification (ws-notification) (specification version 1.0), January 2004.

[2] Arjuna, BEA Systems, Hitachi, IBM, IONA, Microsoft. Web services coordination (ws-coordination) (specification version 1.0), August 2005.

[3] BEA Systems, Computer Associates, IBM, Microsoft, TIBCO Software. Web services eventing (ws-eventing), August 2004.

[4] Hewlett-Packard. Web services events (ws-events) (version 2.0), July 2003.

[5] Frank Leymann and Stefan Pottinger. Rethinking the coordination models of ws-coordination and ws-cf. In *ECOWS '05: Proceedings of the Third European Conference on Web Services*, page 160, Washington, DC, USA, 2005. IEEE Computer Society.

[6] Mark Little and Jim Webber. Introducing ws-coordination. *Web Services Journal*, May 2003.

[7] P. Niblett and S. Graham. Events and service-oriented architecture: the oasis web services notification specifications. *IBM Syst. J.*, 44(4):869–886, 2005.

[8] Oasis. Web services context (ws-context), August 2006.