Towards a Telecommunication Service Oriented Architecture

Paolo Falcarin Jian Yu
Politecnico di Torino, Italy
paolo.falcarin@polito.it , jian.yu@polito.it

Abstract

Web Services are often used for providing and composing business services but this approach does not scale easily for telecommunication services and for value added ones, composing services offered by IT providers with telecom operators ones.

The typical request-response interaction style is the main bottleneck when applying web services protocols to the telecom domain, which requires higher performances and needs efficient ways to handle notifications of events produced by different network resources in the telecommunication infrastructure.

This paper evaluates benefits and drawbacks of Web Service applications within a Telecom domain, analyzing current standardization proposals for asynchronous web services, which are a necessary evolution towards a fully interoperable telecommunication service oriented architecture.

1. New Requirements for Telecom Service Layer

Telecommunication services and network features are often tightly coupled, separate, and vertically integrated. In this context, at service layer there are almost no shared services, specifically because each network has its own vendor proprietary services platform.

This vertical approach has an extremely weakening effect on service provider's ability to develop more complex services that could span over heterogeneous networks and IT services.

In the last years, more complex communication services has been built combining a set of telecom features (call control, WAP gateway, SMS, etc.) exported by more abstract standard interfaces to be used by a software "service layer"; this approach was supported by a rich set of standard API proposals, like Parlay and JAIN, focused on abstracting technical details of the underlying networks (e.g. SIP, PSTN, GSM).

While this was an important step, the integration of communication services, content-based services, Internet-like services, and messaging services has still remained an open problem.

In order to get a fast and flexible integration of communication services, telecommunication operators are now investing effort on the Service Layer, which is devoted to the creation, execution and management of Telecommunication Services, and to fulfill new goals, namely:

- Easy new service creation processes;
- Reduce service creation time and consequent time to market, and costs;
- Evolve the service layer to easily integrate heterogeneous services coming from IT providers with telecom operators ones.

While convergence of heterogeneous networks has mainly succeeded, convergence of heterogeneous services still requires several effort, because of the tight coupling between the service layer and the underlying network infrastructures.

The common vision for implementing services is now the realization of an horizontal service platform, based on shared services and network enablers, which can be easily composed with a loose coupling.

This integration can be addressed by Web Services standards, but their introduction in the Telecom domain means facing up with some problems [3], because communication services have strong real-time requirements (e.g.

high throughput, low latency time), support mainly asynchronous interactions (e.g., voice-mail, call forwarding), and leverage efficiently on native protocol capabilities.

As a consequence the Enterprise Service Bus is emerging as a service-oriented infrastructure component that makes large-scale implementation of the SOA principles manageable in a heterogeneous world, like the telecommunications one.

In next sections we describe what is an Enterprise Service Bus (ESB) and we discuss what is needed to implement it in a telecommunication service oriented architecture.

2. Enterprise Service Bus

In computing, an enterprise service bus (ESB) refers to a software architecture construct, implemented by technologies found in a category of middleware infrastructure products, usually based on standards, that provides foundational services for more complex architectures via an event-driven and standards-based messaging engine (the message bus).

An ESB generally provides an abstraction layer on top of an implementation of a messaging system.

ESB does not implement a service-oriented architecture (SOA) but provides the features with which one may be realized. Although a common belief, ESB is not only web-services based: it typically uses XML as the standard communication language, and it supports web-services standards, but an ESB should be standards-based and flexible, supporting many transport protocols.

More important, a ESB is usually independent from operating-system and programming-language: for example, it should enable interoperability between Java and .NET applications.

An ESB offers messaging with different interaction styles (synchronous, asynchronous, publish-subscribe), and features like queuing, and storing messages if services are temporarily unavailable.

It includes support for service orchestration and choreography, and to ease the transformation of heterogeneous data, it includes transformation services (often via XSLT) between the format of the sending application and the receiving application.

The main benefits of ESB usage are a faster and cheaper integration of existing systems, and an increased flexibility to requirements changes, in order to scale, if needed, from point-to-point solutions to a distributed bus topology.

ESB has some drawbacks, like the overhead due to extra translation layer when compared to regular messaging solutions, and complexity of configuration.

In particular, when ESB are built on a web service infrastructure, the frequent changes typical of recent standards, like asynchronous web services ones, can impact on standard compliance of ESB implementations.

3. Towards an ESB for Communication Services

The main idea of a telecommunication service oriented architecture is depicted in figure 1, where an Enterprise Service Bus acts as a broker of messages among services, using XML for data representation.

The ESB should consent asynchronous interactions among common services in IT infrastructures (like User Profile database, authentication and mail servers, business logic exposed as web service) and typical resources connecting different kind of telecom networks (like SIP for voice over IP and messaging, SMS/MMS gateways for messaging with mobile terminals, resources for handling voice calls with classic PSTN network, and UMTS related protocols for video-calls and advanced connectivity).

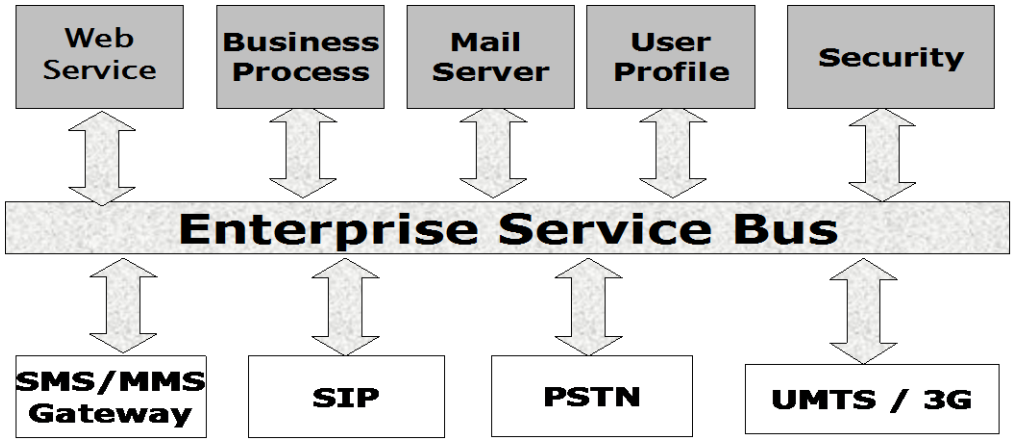

Figure 1. ESB in Telecom SOA

As Web Service standards are becoming widespread for interconnecting IT services, in order to use an ESB relying on web services, different requirements emerge in the telecom domain:

1. The adoption of Service Oriented Architecture (SOA) principles and techniques, to organize the internal structure of the service layer;
2. Adoption of Web Services technologies to expose on the "Internet" communication service components to 3rd party applications;
3. Adoption of Web Services technologies to interact with terminal applications;
4. Evolution of composition mechanisms for creating complex convergent services by assembling basic services and telecommunication features.

Different international consortia have proposed web services standards in order to meet the above mentioned requirements:

- ParlayX [26] has been standardizing WSDL interfaces for the most common signaling services, like Third Party Call, Call Notification, SMS, MMS, etc..
- Open Mobile Alliance (OMA) [27] has proposed the OMA Web Services Enabler to define the means by which OMA applications can be exposed, discovered and consumed using Web Services technologies.
- W3C is providing WS-Addressing specification [14], which promotes asynchronous message exchanges, and allows more than two services to interact.
- OASIS consortium has defined a set of specifications like WS-Notification [10], WS-ResourceFramework [13], and WS-Reliability which are important basis for developing communication Web Services.

At the service layer, the main problem is that, currently, many services have implemented a synchronous communications model, based upon the request/response interaction pattern, where a service requestor identifies a service that it wishes to use and then sends a request message to it, waiting for the response. A second entity, the service provider, accepts the request message, processes it, and then sends a response message. This pattern is used in current Web services implementations.

What is needed for Web Services is an event-based architecture, like the one provided in publish/subscribe systems available from message-oriented middleware.

In contrast to the request/response pattern where the request and response messages are frequently hidden from the programmer, in event-based programming the event has a primary role.

Services explicitly produce and consume events, and the producing service has a relationship with the event that it produces, rather than a direct relationship with the service that consume the event. A consumer of events indicates (through a subscription process) the events in which it is interested, and it interacts with the event itself, rather than with the service that produced the event.

Thus, the notification pattern involves registration of consumers and following dissemination of events.

The publish-subscribe model is useful for integrating one system to many systems. Instead of directly hardwiring one system to another, the destination in this model is a topic to which receivers can subscribe; a receiver is free to subscribe as many topics as desired.

The typical example of this model is a news-service, where the user would like to be notified in real-time about news of interest.

Next section will analyze recent standard proposals and implementations for realizing publish-subscribe model in web services domain, and for implementing communication web services with asynchronous interactions.

4. Web Services protocols for Asynchronous Interactions

In a publish-subscribe architecture, interaction among Web-services is realized by means of messages exchange. A message contains may contain various kind of information, like application data, errors, and results of service discovery.

Currently, a couple of web services directly interact by means of basic request/response interaction pattern.

Instead of one-to-one interactions, the publish-subscribe architecture promotes interactions of one-to-many type, through multiple dispatching of event notifications, allowing a real separation among the message sender and one or more recipients.

Dispatching of events is not broadcast but an event consumer must declare which events it is interested to.

This event subscription phase can be either managed by the event-producer itself (which in turn sends the event notification to all subscribed consumers) or be delegated to a third party (broker) which is responsible for dispatching event notifications to event consumers: in this case the subscription to events of interest must be also managed by the broker.

Moreover, a discovery phase is always needed to find which services can notify events. Once these services are found it is possible to ask for information on events which are observable.

After discovery, a subscription phase is required from event-consumers to define which events they want to be notified of.

Subscription enforces event-producer to notify events to the consumer. The subscription can have limited time validity and in this case it can be renewed.

Once subscription is completed, producers must notify all event consumers, i.e. it pushes event to all consumers.

Asynchronous Service Access Protocol (ASAP) [28] was an initial tentative for handling asynchronous interactions. ASAP is a protocol for extending the request-response interaction model of SOAP with the possibility to deal with requests with long computations; in this case the results are returned on a Web Service interface provided by the invoking application (according to a call-back model); ASAP also defines the former interfaces and defines other ones which provides operations to monitor the execution status.

The main drawbacks of this approach are two: the request parameters and the results are returned as a “blob”, so loosing all the information concerning the typing of the interfaces; it does not provide a native support to the handling of multiple notifications, i.e. the publish/subscribe model.

Currently, more recent specifications exist for supporting publish/subscribe model in web services domain: WS-Events [8], WS-Eventing [9], WS-Notification [10] that are detailed in next sections.

4.1 WS-Events

Web Services Events (WS-Events) [8] is a standard proposed by HP, crafted to introduce the above-mentioned publish/subscribe scenarios in Web Services standards.

WS-Events defines an “event” as a notification of the change of the state of a resource.

An event can be represented by an XML element called “notification”, which can be generated by an “event producer”. An event producer can emit several event notifications, in turn received by one or more event consumers.

WS-Events defines a simple subscription protocol among event consumers and event producers, and it specifies subscription mechanisms.

Event notifications can be dispatched by means of synchronous communication (“pull” mode) or asynchronous one (“push” mode).

In push-mode the event producer invokes a callback method on an event consumer, with a set of notifications as parameter. In practice, the producer asynchronously communicates notifications to the event consumer.

In pull-mode, instead, the consumer invokes methods on producer to recover the notification messages generated from the previous call.

In this case the producer must store notification messages, which have not yet been requested by the subscribed event-consumers.

Moreover, WS-Events provides a lightweight XML syntax for defining event filters, which can be declared during subscription phase.

WS-Events does not aim at defining a general-purpose event-based mechanism; it represents one of the possible publish/subscribe mechanisms for web services. WS-Events relies on WSDL [7] and SOAP [6] and the current specification completely lacks of the definition of a broker, which is very important in several event-based systems. Moreover, WS-Events does not face with specifying service discovery mechanisms, that could be used by event consumers to find out event producers.

As a consequence, discovery is triggered by the consumer which has to query the producer for discovering events, then it must communicate to the event producer its interest in receiving notification of a particular event.

Subscription message contains the event type and optionally the URL of a callback method, which will be called by the producer to notify the occurrence of an event of the desired type. If a callback is not specified, then the consumer has to obtain events from the producer (pull-mode subscription).

During subscription phase the consumer can specify a "lease" parameter, which represents the time validity of the requested subscription. Each subscription has a unique id and a lease timestamp.

Therefore a subscription message contains:

- A selector of event types that must be notified to the consumer;
- A proposed lease time of the subscription;
- An optional event filter, e.g. a XML document, or an XPath [15] string;
- An optional callback method to be invoked on the consumer interface.

WS-Events does not define neither the syntax nor the semantics of a filter, which must be agreed between producer and consumer.

4.2 WS-Eventing

The WS-Eventing [9] standard was proposed by an industrial consortium (created by BEA Systems, Computer Associates International, IBM, Microsoft, Sun Microsystems, and TIBCO Software).

WS-Eventing defines a simple protocol for handling event subscriptions among web services.

In WS-Eventing an event occurs whenever the state of a resource changes. WS-Eventing defines three roles: subscriber (or sink), event source, and subscription manager.

A subscriber is a web service which wants to receive event notification messages. In order to meet this goal, the subscriber registers to events of interest on another web service, the event source. The event source behaves like the notification producer defined in WS-Notification specification. Subscriber can handle subscriptions interacting with a web service (subscription manager), designated by the event source.

WS-Eventing aims at defining a protocol for creating and deleting event subscriptions, handling with their limited validity in time and with their renewal, and at defining how an event source can delegate subscription management to a third party (the subscription manager).

WS-Eventing relies on WSDL, SOAP, and WS-Addressing, but it does not face with the issue of discovery of event sources.

The subscriber (sink) behaves exactly like the event consumer defined in WS-Events specification.

The sink has the role of subscriber and notification consumer.

4.3 WS-Notification

WS-Notification [10] defines a set of specifications that standardize the way Web services can interact using the Notification pattern, which defines a way for consumers to subscribe to a producer for notifications whenever a particular event occurs.

The WSN family is made up of four separate specification documents.

The WS-Base Notification specification defines the Web Services interfaces for notification producers and notification consumers. It includes standard message exchanges to be implemented by service providers that wish to act in these roles, along with functionalities expected of them. This is the base document on which the other WSN specification documents depend.

The WS-Topics [18] specification defines a mechanism to organize and categorize items of interest for subscription known as topics. A topic is a type of events which can receive subscriptions for notifications.

Comparing with object-oriented programming, a topic can be seen as a class, because it has properties (attributes) and methods for querying its internal state. These are used in conjunction with the notification mechanisms defined in WS-Base Notification [16].

WS-BrokeredNotification [18] defines the Web Services interfaces for notification brokers, when message broker is present among notification producers and consumers.

WS-Notification defines interfaces and behavior of five possible roles:

- Notification Producer;
- Notification Consumer;
- Subscriber;
- Subscription Manager;
- Notification Broker.

The Notification Producer receives subscription requests from notification consumers. Each subscription request identifies one or more topics of interest for the Notification Consumer. Each subscription contains:

- Topic (on which the subscription has been made);
- The Id of the Notification Consumer interested to events related to the specified topics;
- Optional information (which semantics is not specified in the specification).

If a broker is not present, the Notification Producer must maintain the subscriptions list and handling notification dispatching.

A producer can delegate the subscriptions list handling to the Subscription Manager, which has to dispatch subscription requests, coming from event consumers, to event producers.

The Notification Broker, if present, can both handle subscriptions and notifications transmissions.

As mentioned before, WS-Notification is composed of three specifications: WS-BaseNotification, WS-BrokeredNotification, WS-Topics:

- WS-BaseNotification defines behavior between notification producer and consumer, and subscription phase whenever there are no brokers.
- WS-BrokeredNotification specifies the interface of notification broker and the protocol with brokered notification.
- WS-Topics defines how to structure the topics in hierarchies.

In case of WS-BaseNotification, the subscriber checks the topics hierarchy exposed by the notification producer.

Once interesting topics are found, the subscriber makes a subscription request to the producer. After subscription, the producer updates the subscriptions list, and delegates a subscription manager the task of handling a particular subscription.

Then the producer must send to all subscribed consumers the notification messages.

Event consumer will receive the events and it may modify the resource state, acting on the related subscription manager. Each entity can cover different roles, i.e. it can implement different interfaces.

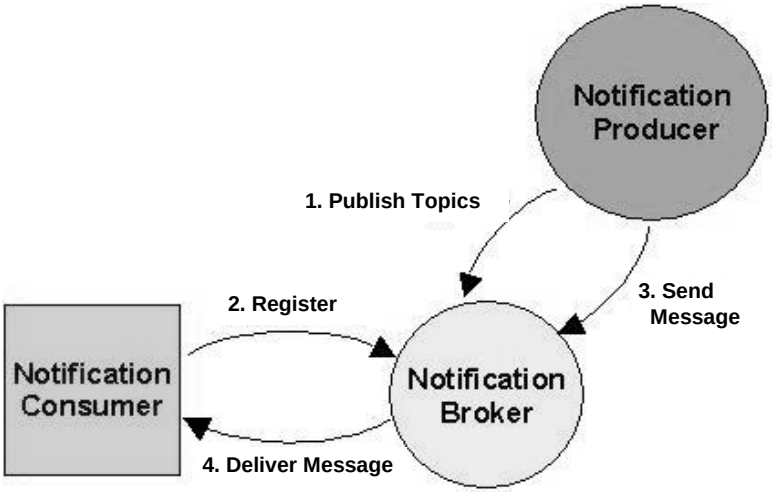

Figure 2: Publish-Subscribe with Broker in WSN.

In figure 2 the basic behavior of brokered notification is illustrated.

The Notification Producer is responsible for specifying the list of topics to which users can subscribe, and topics can be added or deleted dynamically. The sequence for the publish-subscribe model is as follows:

1. The Notification Producer communicates to the broker the topics to which the subscribers can subscribe.
2. The subscriber registers its own callback (a message handler to be invoked when a message is dispatched) to the broker.
3. The Notification Producer creates a message related to a topic and sends it to the broker.
4. The broker delivers the message to the appropriate subscribers, thereby invoking the appropriate callback of each subscriber. Each subscriber is then free to perform its own processing when it receives the message.

Once all the subscribers have been notified and each subscriber has acknowledged the receiving of a message, the integration broker removes the message from the topic.

This model provides for durable subscriptions, which is suitable for clients that are often offline, thus it is a good model for wireless devices, which cannot be constantly connected. The integration broker stores messages persistently and forwards them to the wireless device when the device comes back on-line.

All subscribers in a topic-based system will receive all messages published to topics which they subscribed to.

Broker has to keep information about topics, and during discovery phase, the subscriber looks for topics of interest: the broker also allows the use of nested topics, which form a hierarchy.

Publisher registers on broker the events it will send about one or more topics. The events will be transmitted only to the broker, which holds a subscription list of all consumers, and then it will dispatch events accordingly.

4.4 Comparison among Specifications

Each one of the above mentioned specifications relies on existing standards; while all rely on SOAP as a message transport protocol, WS-Eventing reuses WS-Addressing [14] to identify Web service endpoints and to secure end-to-end endpoint identification in messages.

WS-Notification specifications set contains WS-BaseNotification, WS-Topics, WS-BrokeredNotification, and it relies on WS-ResourceFramework specifications.

Main features of the three standard proposals are depicted in the following table.

Common lacks of these three specifications are:

- Management of reliable messaging
- Definition of timeouts on notification request when working in pull-model;
- Subscriptions Management policies are not customizable.

- Discovery of Notification Producers.

The strong limitation of WS-Events and WS-Eventing is the lack of a brokered notification, while WS-Notification is the only specification using topics.

	WS-Notification	WS-Events	WS-Eventing
Broker	Yes	No	No
Notification model	Push	Push, Pull	Push, Pull
Subscriptions delegation	Yes	No	Yes
Topics	Yes	No	No
Topics Advertisement	Yes	No	No
Events hierarchies	Yes	No	No
Subscription renewal	Yes	Yes	Yes
End of subscription notification	No	No	Yes

Table 1. Asynchronous WS Standards

5. Apache Muse

Apache MUSE [21] is a Java implementation of the Management Using Web Services (MUWS) 1.0 specifications [22], which has included the initial Apache PubScribe project [12], implementing WSN specification. The latest MUSE version 2.2 integrates the Java implementation of three standards: WSRF 1.2, WSN 1.3 and WSDM 1.1 [23]. It is a framework upon which users can build web service interfaces for manageable resources using a high level API and without worrying about the details of the aforementioned standards. Applications built with Muse can be deployed in both Apache Axis2 [24] and OSGi [25] environments, and the project includes a set of command line tools that can generate the proper artifacts for deployment of this new kind of web services.

WSRF defines a generic and open framework for modeling and accessing stateful resources using Web services, where a resource is a generic concept used for representing events, devices or software components.

This includes mechanisms to describe views on the state, to support management of the state through properties associated with the Web service.

Muse provides a WSDL to Java generator that will generate the classes and configuration entries from a WSRF/WSN WSDL that are required to deploy it as a service to the Apache WSRF framework.

Moreover, Apache WSRF offers full implementations of all portTypes defined by the WS-Eventing (WSE) specification, and it offers a generic Publish/Subscribe API that hides the details of the used notification mechanism (WSN or WSE).

Muse allows publishing stateful web services, which expose topics, and accept subscriptions by means of SOAP messages: whenever a topic's state changes, notification SOAP messages are sent to the event consumers.

Muse defines an API for exposing a WSRF resource as a topic, in order to be observed by subscribed clients.

In case of a news-service Muse can be used to broadcast news messages to all subscribers.

In case of more complex communication services, this behavior can be customized using event filters, because topics, in general, are independent from users' data.

6. Conclusions

In this paper we described current standardization efforts for asynchronous web services in order to obtain a communication service oriented architecture; among these, WS-Notification is the more complete set of standards, and its open-source implementation, Apache Muse is currently a viable way for implementing communication web services, based on publish/subscribe asynchronous interaction pattern.

In particular, WS-BrokeredNotification specification can be implemented by Enterprise Service Bus for telecommunication service oriented architectures.

In this interaction model publishers are loosely coupled to subscribers, and need not even know of their existence, as the message dispatching and service discovery is delegated to the ESB. With the topic being the focus, publishers and subscribers can be unaware of system topology. Each one can continue to operate normally regardless of the other. In the traditional tightly-coupled client-server paradigm, the client cannot post messages to the server while the server process is not running, nor can the server receive messages unless the client is running.

Enterprise Service Bus provides the opportunity for better scalability than traditional client-server, through parallel operation, message caching, brokered notifications, thus it is the main challenge for achieving a real reliable and interoperable telecommunication service oriented architecture.

7. References

[1] C.A. Licciardi, and P. Falcarin, "Analysis of NGN service creation technologies", Annual Review of Communications vol. 56, IEC, 2003, pp 537-551.

[2] C.A. Licciardi, and P. Falcarin, "Next Generation Networks: The services offering standpoint". In Comprehensive Report on IP services, Special Issue of the IEC, 2002.

[3] A. Baravaglio, C.A. Licciardi, C. Venezia. "Web Service Applicability in Telecommunications Service Platforms". In Proc. of the International Conference on Next Generation Web Services Practices, Seoul, Korea, August 2005

[4] XML (eXtensible Mark-up Language) specification. On-line at http://www.w3.org/XML/

[5] "Introduction to UDDI: Important Features and Functional Concepts", OASIS, October 2004

[6] SOAP (Simple Object Access Protocol) specifications. On-line at http://www.w3.org/TR/soap/

[7] WSDL (Web Service Definition Language) specification. On-line at http://www.w3.org/TR/wsdl

[8] WS-Events specification. On-line at http://devresource.hp.com/drc/specifications/wsmf/WS-Events.jsp

[9] WS-Eventing specification. On-line at http://www-128.ibm.com/developerworks/library/specification/ws-eventing/

[10] Web Service Notification specifications. On-line at http://www.oasis-open.org/committees/wsn

[11] AXIS project. On-line at http://ws.apache.org/axis/

[12] Apache Pubscribe project. On-line at http://ws.apache.org/pubscribe/

[13] Web Service Resource Framework specifications. On-line at http://www.oasis-open.org/committees/wsrf

[14] WS-Addressing specification. On-line at http://www.w3.org/Submission/ws-addressing/

[15] XPath specification. On-line at http://www.w3.org/TR/xpath

[16] WS-BaseNotificaiton specification. On-line at http://docs.oasis-open.org/wsn/wsn-ws_base_notification-1.3-spec-os.pdf

[17] WS-BrokeredNotification specification. On-line at http://docs.oasis-open.org/wsn/wsn-ws_brokered_notification-1.3-spec-os.pdf

[18] WS-Topics specification. On-line at http://docs.oasis-open.org/wsn/wsn-ws_topics-1.3-spec-os.pdf

[19] WS-ResourceProperties specification. On-line at http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties-1.2-draft-04.pdf

[20] WS-ResourceLifetime. http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceLifetime-1.2-draft-03.pdf

[21] MUSE. On-line at http://ws.apache.org/muse/

[22] MUWS. On-line at http://www.oasis-open.org/committees/download.php/17000/wsdm-1.0-muws-primer-cd-01.doc

[23] WSDM. On-line at http://www.oasis-open.org/committees/wsdm/

[24] AXIS2. On-line at http://ws.apache.org/axis2/

[25] OSGi. On-line at http://www.osgi.org/

[26] ParlayX. On-line at http://www.parlayx.com/

[27] OMA (Open Mobile Alliance). On-line at http://www.openmobilealliance.org/

[28] ASAP protocol specification. On-line at http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=asap