Web Services-based Middleware for QoS Brokering of Media Content Delivery Services

Sofie Van Hoecke, Kristof Taveirne, Koen De Proft, Filip De Turck, Bart Dhoedt
Department of Information Technology
Ghent University, Ghent, Belgium
fax: +32 9 264 9960 - tel: +32 9 264 9970
sofie.vanhoecke@intec.ugent.be

Keywords: Web services, multimedia, broker, dynamic service selection, service composition, performance.

Abstract—**Powerful computers and affordable digital media appliances result in a rapid growth of possibilities in the field of multimedia applications. There has been a significant increase in creation and use of multimedia content. Nowadays everybody can be treated as a potential creator and user of multimedia content. In order to deliver multimedia services to end-users, many parties are integrated and a series of Web services is accessed. A Web service in the composition can become unreachable by a number of circumstances, which requires run-time actions to replace it by another compatible Web service. A Web service can also perform less well than guaranteed, which also requires action in order to fulfill QoS requirements. To answer these requirements, we designed a broker middleware offering dynamic selection and composition of Multimedia Content Delivery services and automatic load balancing of these services over the available application servers. Since users want fast and reliable connections and they do not really care what is happening in the background, the broker platform transposes all services in a transparent and similar fashion, hiding for the users if it concerns a simple or composed service and thus simplifying required user interactions. The broker architecture intercepts all service requests and makes sure those requests are redirected to the best provider server, even in the case of composed services. The architecture will also allow the automatic fulfillment of QoS requirements, such as response time, availability, quality and price.**

I. INTRODUCTION

With the rapid growth of possibilities in the field of multimedia applications, there has been a significant increase in creation and use of multimedia content. Nowadays everybody can be treated as a potential creator and user of multimedia content. In order to deliver multimedia services to end-users, many parties are integrated and a series of Web services (authentication, billing, streaming, etc.) is accessed, which are either not or only partially included with the actual application. Two problem dimensions can be distinguished here. On the one hand is the problem of security. End-users who pay for multimedia services want to be sure that there are no privacy leaks and that the content is delivered to them in a secure way. Therefore, the author presented in [1] an architecture for secure Media Content Delivery (MCD) using the combination of both SSL and WSS. Since however many parties are integrated in advanced MCD services, users want to reduce their trusted relationships. On the other hand is the problem of fulfilling QoS requirements. A Web service

in the composition can become unreachable by a number of circumstances, which requires run-time actions to replace it by another compatible Web service. A Web service can also perform less well than guaranteed, which also requires action in order to fulfill QoS requirements. Both problems, reducing trust relationships and fulfilling QoS guarantees, were not addressed in [1] and has been the subject of recent research. In this paper, we detail the design of a broker architecture that allows for dynamic selection and composition of services, and simplifies required user interactions.

Integration of multimedia services and applications is however a complex task since these services and applications are built by different vendors, using different data definitions and exchange standards. Here Web services can gain more and more importance for the integration of heterogeneous MCD components due to their standardised interoperability and compatibility with other languages.

A. Web service technology: State-of-the-art

In view of the broad support for Web services and common XML-based standards such as SOAP, WSDL and UDDI, Web services are a promising concept for the integration of heterogeneous software components. By means of this technology, applications can easily be distributed over the Internet and expose well-defined functionality as a Web service, which consumes and produces XML-messages over HTTP. Based on the exchange of structured text messages, the interaction abstracts the underlying technologies. Consequently, extending existing services with a Web service interface enables integration [2]. Since Web services can provide overlapping or identical functionality, albeit with different QoS, a choice needs to be made to determine which services are to participate in a given composition. Web service composition however has a twofold meaning. On one hand composition in the sense of granularity consists of encapsulating Web services in a larger part and exposing this as a Web service. On the other hand composition in the sequencing sense defines the invocation order of Web services. Thus composition consists of those activities required to combine and link existing Web services to create a variety of new and complex processes.

There are however many existing approaches to service composition, ranging from abstract methods to industry standards. Service composition is still not standardized, nor does it include definitions of the key requirements that every composition approach must satisfy (such as scalability, dependability or correctness). The first-generation composition languages WSFL and WSCI were incompatible and therefore second-generation languages, such as BPEL4WS, were developed combining WSFL and WSCI with Microsoft's XLANG specification.

Business Process Execution Language for Web Services (BPEL4WS) [3] is an XML language that is currently being standardized by OASIS and which supports process-oriented service composition. BPEL4WS provides a language for the description of business processes and transactions, supporting both orchestration and choreography. On one hand, in orchestration a central process takes control of the involved services and coordinates the execution. On the other hand, choreography does not rely on a central coordinator. Each Web service involved knows when to execute and with whom to interact. Within BPEL, executable processes allow you to specify the exact details of business processes and provide the orchestration support, while the abstract business protocols allow specification of the public message exchange between parties only, thus focusing more on Web services choreography. BPEL4WS is built on top of XML, XML Schema, WSDL and UDDI, and is the most well established orchestration technology for Web services.

Web service flow specification languages, such as BPEL4WS, describe in which order messages have to be exchanged between services in a flow specification. The Semantic Web community developed however their own flow specification languages using semantic annotations where preconditions and effects of services are explicitly declared in the Resource Description Format (RDF) using terms from pre-agreed ontologies. Thus while BPEL4WS is more business oriented, it is also possible to composite services with the Semantic Web alternative OWL-S.

Using OWL-S [6], a client application could discover a sequence of independent services that could ultimately help it to satisfy its information goal. Since dynamic composition of services is not yet provided by OWL-S, an additional proprietary AI planner must be implemented at the client level. OWL-S simply provides for the discovery of services, not their automatic composition. Automatic composition would require more planning logic at the application layer in order to enable the applications themselves to compose collections of services into a more complicated operation. However, OWL-S does support the manual composition of services into composite processes to solve a predefined goal.

Both BPEL4WS and OWL-s are not fundamentally different but have a different approach, as well as their advantages. BPEL4WS is strong on orchestration and info sharing, while OWL-S is strong on goals, activities and discovery. With regard to Web service composition, both solutions will need to co-operate in order to ensure that one standard will eventually emerge for the all-over composition of Web services.

However QoS (Quality of Service) requirements such as response time, availability and price can also influence and hamper service composition. Therefore, in this paper, we present a middleware platform for brokering of composed services with QoS guarantees. Implementing this brokering middleware by means of Web service technology creates the required integration of heterogeneous services, taking into account QoS requirements, and offers an advanced set of composed services crossing multiple service and data providers. This way the required user interaction is reduced to authenticating (e.g. by means of eID, smart card or login/password) and selecting one of the (composed) services.

B. Application cases

The presented architecture covers a wide range of application cases. For example eCommerce can benefit from QoS brokering if a call center for example negotiates with multiple credit checkers, in order to acquire payment validation. Based on the call center load, the broker middleware can divide the requests over multiple credit checkers in order not to lose or displease clients. Another case can be found in eHealth where integrated and composed eHomeCare services (eRecord, ePrescription, teleMonitoring, etc.) can cross health care providers and medical databases. QoS brokering can also be applied in B2B (Business to Business) for optimizing the virtual supply chain of delivery companies. QoS brokering can select delivery services with the best QoS (e.g. delivery time, quality, price), resulting in smaller stocks and advanced efficiency. On the other hand, QoS brokering can also be useful in pure software design, planning and measuring the required infrastructure, as well as in offline tuning of load balancing and brokering strategies. Finally Multimedia Content Delivery is another case that can benefit from QoS brokering. The broker middleware can dynamically select and compose the needed services (e.g. services for broadcasting, streaming, payment and security) in order to set up for example a video-on-demand stream meeting the request (e.g. high quality, no delay or limited output device). The deployment of the QoS brokering middleware for MCD is described in this paper.

C. Related work

Related work in this area focuses mostly on Web service brokers limited to service lifecycle management. There is however a need for service brokers taking into account QoS in order to ensure total response time of composed services, prioritize time-critical services or ensure bandwidth or robustness. In [8] a QoS broker model is described for general distributed systems. Contrary to general distributed systems, Web services have a dynamic nature in terms of service availability and the clients invoking them. Brokers must support more flexible service selection and be able to adapt to the dynamic server load. In [9] a Web service architecture supporting QoS is presented. However, once the services are selected and the link is established, the client communicates with the server directly without any

broker intervention during the actual service process. Due to the dynamic nature of Web services, this introduces QoS shortcomings since abrupt failure or unavailability of services needs dynamic selection of another equivalent service. This broker also leaves composition as the client's responsibility.

The remainder of this paper is structured as follows: Section II describes our QoS brokering middleware, while in section III some performance results are presented. Finally, in section IV, we will highlight the main conclusions and identify some future work.

II. QoS BROKERING MIDDLEWARE

Figure 1 depicts the open QoS broker architecture, presented in this paper.

As can be seen in this figure, the architecture contains three different levels of stakeholders. Authentication can be provided by a third party. Next to this, there's a broker domain and several provider domains. If the interfaces to the broker domain are respected, other legacy systems in the provider domains are possible, such as Mosix [10] or Unicore [11]. However we believe in the simplicity and openness of our presented solution for the provider domains.

First the main functions of the architecture are described after which a detailed description of each component is given.

A. Functional description

The architecture is built around a facade portal. This portal contains a UDDI (Universal Description, Discovery and Integration) registry containing all the services offered by the platform, i.e. the simple services offered by the service providers, as well as advanced composed services only possible by the platform. The platform however transposes all services in a transparent and similar fashion, hiding for the users if it concerns a simple or composed service and thus simplifying required user interactions (see figure 2a).

This portal intercepts all service requests and makes sure those requests are redirected to the best provider server, even in the case of composed services. Since the broker platform has to be able to interpret the service request, the requests are transformed into a composition flow chart (see figure 2b). In order to select the best suited servers for fulfilling the request, the middleware queries for interfaces of all the possible matching services. After filling in this information into the flow charts (see figure 2c), the middleware requests for the appropriate QoS metrics. By using advanced broker algorithms, the best path through the flow chart is chosen. Through this flow processing, service sessions are assigned to the best suited provider server, based on the capabilities of the server (which service(s) does it implement) and its QoS characteristics such as load, availability, service price and the already running services. This flow processing is described more in detail in section 4. Afterwards, the middleware platform aggregates the result sets, translates this flow response into a service response and returns it to the requester.
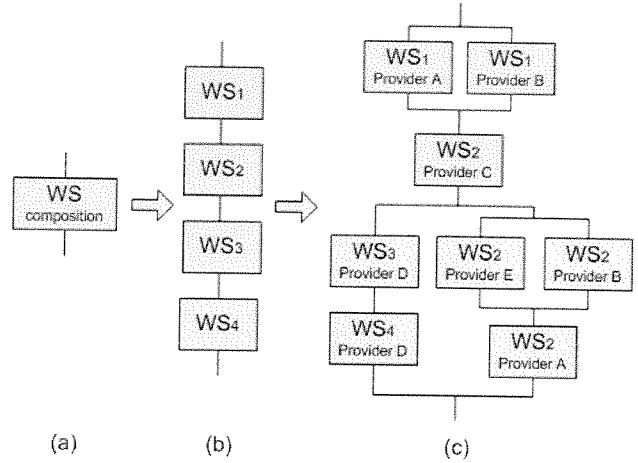

(a)   (b)   (c)

Fig. 2. Service composition flow: (a) Transparent services offered by the Broker Facade (b) Composition flow chart (c) Matching services for composition

B. Component description

The architecture consists of the following components:

- **Broker Facade (BF):** The Broker Facade forms the central portal of the brokering middleware. It intercepts incoming service requests from authenticated users, requests needed credentials from the Credential Manager (CM), adds them to the service requests and forwards these requests to the Mapping Service Flow (MSF) component.
- **Credential Manager (CM):** The Credential Manager contains user credentials based on the registration input or profile. Incoming service requests from the Broker Facade will be adjusted with these credentials before being forwarded to the Mapping Service Flow component.
- **Mapping Service Flow (MSF):** The MSF translates incoming service requests into corresponding flow charts and forwards these flows to the Flow Processing Service (FPS). Incoming flow responses are translated back to service responses before being forwarded to the Broker Facade.
- **Flow Processing Server (FPS):** The Service Directory Repository (SDR) returns the Application Servers (AS) offering the requested services. Based on the QoS information provided by the Pool Management Server Coordinator (PMSC), the Flow Processing Server can select, by using advanced broker algorithms, the optimal Application Servers for executing the services, fulfilling the QoS requirements.
- **Service Directory Repository (SDR):** The Service Directory Repository is implemented by exploiting the UDDI technology, a framework for the description and discovery of services. The architecture contains one or more linked Service Directory Repositories. Here all services offered by the Application Server are registered
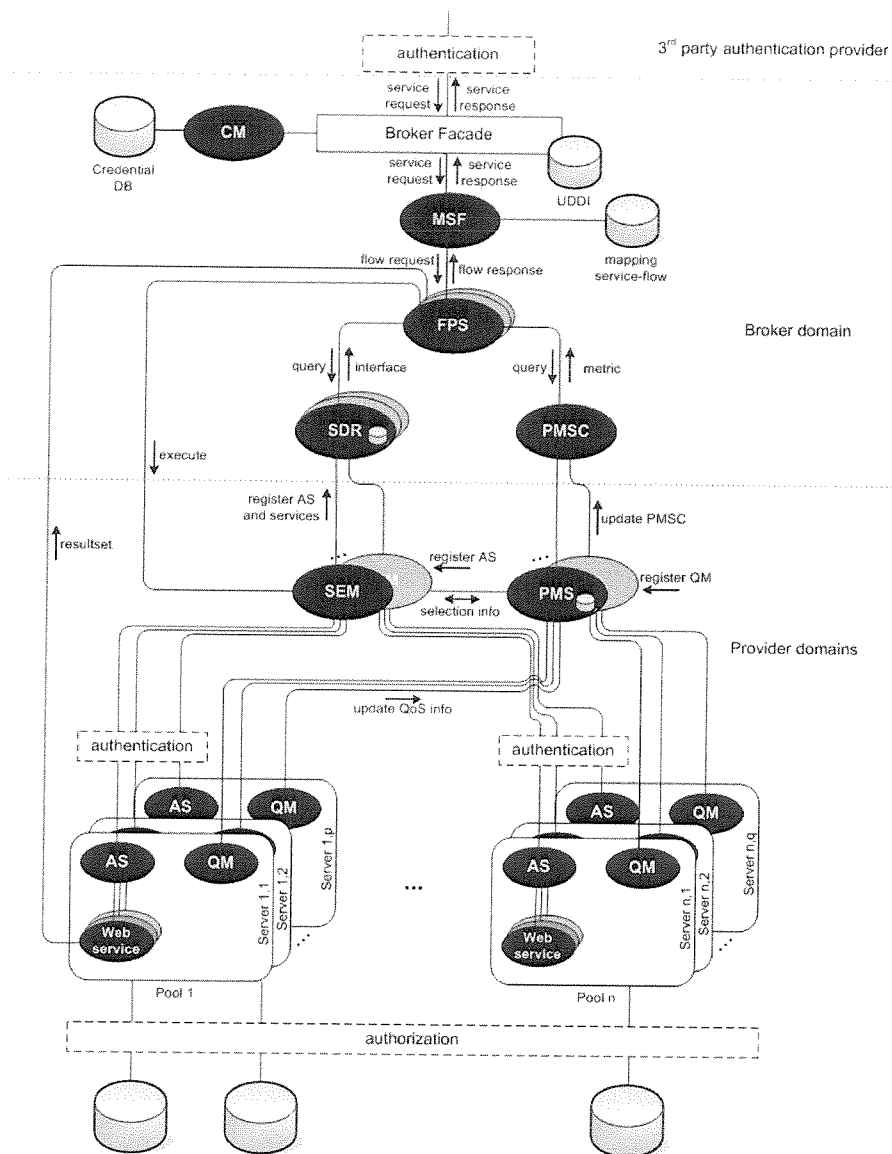
Fig. 1. QoS brokering middleware platform. The following components are shown: Credential Manager (CM), Broker Facade (BF), Mapping Service Flow (MSF), Flow Processing Server (FPS), Service Directory Repository (SDR), Service Execution Manager (SEM), Pool Management Server Coordinator (PMSC), Pool Management Server (PMS), Application Server (AS) and QoS Monitor (QM).

and described in detail (location, business properties, service properties, etc). The Service Directory Repository contains also the Service Level Agreements (SLAs), containing the agreed capability information such as service availability, maximum response time, price, etc. When a new Application Server is added to a pool, the Service Execution Manager (SEM) registers this server and his services to one of the Service Directory Repositories. Since all Service Directory Repositories are referencing each other, it appears like one single SDR. Creation and deletion of pools, as well as registering a new Application

Server to the Service Execution Manager of a certain pool, is done through the Pool Management Interface of the Service Directory Repository. When a new pool is created, both an Service Execution Manager (SEM) and a Pool Management Server (PMS) for that pool are started.

• **Service Execution Manager (SEM):** The Service Execution Manager manages a pool of Application Servers. When a new Application Server (AS) is started, it has to register itself to the SEM of that pool. The SEM then registrates that Application Server and its Web services to the Service Directory Repository (SDR). When new

Web services have to be started on specific Application Servers, the FPS contacts the SEM of the pools these ASs belong to. The SEM then forwards the start commands to the correct AS. Result sets from these Application Servers however are not sent back to the SEM, but directly forwarded to the FPS.

- **Application Server (AS):** Application Servers execute the actual Web services. Each Application Server can implement one or more Web services, can have its own capabilities and can provide services using different programming languages. When a new AS is installed in the system, it must register itself to the Service Execution Manager (SEM) of the pool it belongs to.

- **Pool Management Server Coordinator (PMSC):** The Pool Management Server Coordinator stores an overview of the QoS information of each server running an AS/QM pair. This information is updated at regular intervals by the Pool Management Server of each pool. At regular time intervals, an overview of the QoS of the platform is pushed from the PMSC to the FPS. When executing the dynamic selection algorithm, the FPS can request more detailed and up-to-date information about each server running an AS/QM pair from the PMSC.

- **Pool Management Server (PMS):** Each pool has its own Pool Management Server. It gathers the QoS information of the monitors in the pool. At regular intervals, the QoS information of the whole pool is reported to the Pool Management Server Coordinator (PMSC). When a new Application Server is started, the QoS Monitor (QM) that it is paired with has to register itself to the PMS of the pool it belongs to.

- **QoS Monitor (QM):** Each Application Server is paired with a QoS Monitor that monitors Quality of Service (QoS) characteristics of the machine it is running on. Different QoS parameters can be measured: CPU load, memory usage, load average, availability, price, etc. At regular intervals the QM reports the QoS of its machine to the Pool Management Server of the pool it belongs to. When a new AS is started, the QM it is paired with has to register to the PMS of its pool.

C. Flow processing

In order to process the composition flows, the FPS component implements advanced broker algorithms for dynamically selecting and composing services, based on the capabilities of the Application Servers and their QoS characteristics such as load, availability, service price and the already running services. The broker uses heuristics for solving the Constrained Shortest Path Problem (CSP) in order to find a path solution and select the required services. Therefore the selection of Web service endpoints, that can provide the requested level of QoS, needs to be made at runtime in order to achieve the best possible execution path of the business process at that particular point in time. Since both BPEL4WS and OWL-S allow us to leave out the hard coded endpoint references of the partners of the business process,

adding this information at runtime when it is needed, the FPS component will support both OWL-S and BPEL4WS standards. Since however commercial solutions are yet to embrace the OWL-S technology, only BPEL4WS processing is currently implemented in the FPS. We are however planning to support OWL-S in future as well.

Within BPEL4WS, the relationships between the partners of a business process are described using partnerLinkTypes. In a conversation between two partners one or each of these partners are assigned a role. This role element refers in turn to a portType defined in an available WSDL file. This WSDL is in fact an abstract description of the Web service, meaning a concrete reference to an endpoint is not defined.

```
<plnk:partnerLinkType name="VoDPartnerLinkType">
<plnk:role name="source">
<plnk:portType name="medianet:VoDPortType>
</plnk:role>
</plnk:partnerLinkType>
```

The BPEL process is a composition of abstract services, meaning that only logical names and services interfaces are used. End-point resolution happens at later stage. A PartnerLink is by this definition strictly a logical construct.

A Partner Link Type points to a WSDL portType with the abstract interface of a Web service. In order to establish the relationship between the abstract interface and the physical deployment of the Web service, the partnerLink needs to be resolved to a concrete endpoint. BPEL uses the concept of endpoint references. These are defined by the WS-Addressing specification and allow a dynamic identification and description of service endpoints. These endpoint references contain on one hand the address of the service endpoint, and on the other hand policies to describe amongst others, the requirements and capabilities of a service endpoint.

```
<wsa:EndPointReference ... >
Xmlns:cen=http://medianet.ibcn.be/services/VoD>
<wsa:Address>http://medianet.ibcn.be/services/VoD
</wsa:Address>
<wsa:PortType>cen:VoDPortType</wsa:PortType>
<wsa:ServiceName PortName="VoD">
Cen:VoD
</wsa:ServiceName>
<wsp:Policy ... >
</wsa:EndPointReference>
```

The WS-Addressing specification standardizes the way Web services are referenced without binding them to a specific transport mechanism. For each partner role defined in a partner link an endpoint type can be defined using a process deployment descriptor file. A first important endpoint type is the Dynamic Endpoint Type. This type indicates that the endpoint reference is provided within the BPEL process. This way the location of the Web service can be assigned dynamically by simply using an Assign activity in the BPEL-flow. A second important endpoint type is the Invoker Endpoint Type. This type indicates that the invoker of the Web services defines what

instances must be used to handle the request. This can be the case when the client or invoker has a preferred Web service instance. Using the dynamic endpoint type, complex path algorithms can be used to calculate the optimal path for the business process execution. Even authentication information that travels along in the SOAP headers of the process flow can be used to make a decision about the endpoint references that should be used for that particular user.

The WS-Addressing specification allows for a WS-Policy section to be embedded in the WS-Addressing EndPointReference block. This means that extra criterias are possible to make a selection between the different available endpoint references based on pure policy-information. Our heuristics for the Constrained Shortest Path Problem use this policy information in order to make a choice by using Pick activity in the process flow.

Supplied with the service interfaces and corresponding service QoS information, the FPS can select the best path through the chart, satisfying the required end-to-end QoS constraints (see figure 3a). However when one of the components suddenly slows down or becomes unavailable, the FPS chooses an alternative path through the flow chart (see figure 3b). This selection and composition process is repeated one or more times until the path is completed.

(a)   (b)
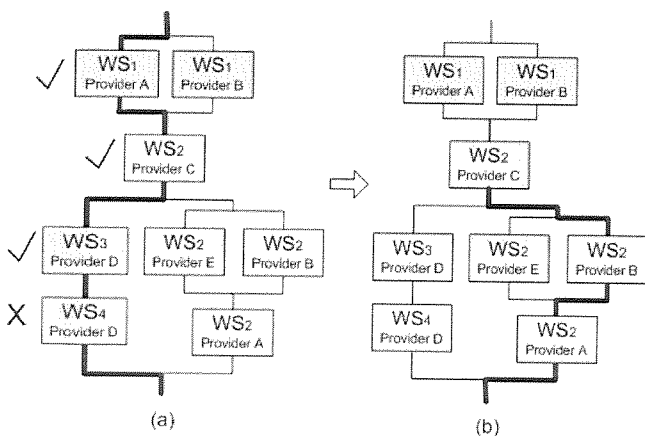
Fig. 3.   Handling service failure

D. Component interaction

Figure 4 depicts the sequence diagram, presenting the component interactions within the middleware platform.

An Application Server is paired with an QoS Monitor (QM), that monitors Quality of Service characteristics (such as CPU load, memory usage, load average and price) of the machine it is running on. The Application Servers are organized into pools. Pools can be created and removed through the Pool Management Interface of the Service Directory Repository (SDR), implemented by exploiting the UDDI technology. Each pool has one Service Execution Manager (SEM) component as well as one Pool Management Server (PMS). When a

new Application Server is installed into the system, it must register itself to the SEM of the pool it belongs to. The SEM consequently registers the AS and its Web services to the Service Directory Repository. When a Web service has to be started on a specific Application Server, the FPS contacts the SEM of the pool that Application Server belongs to. The SEM then forwards the start service command to the correct AS, taking into account selection information from the Pool Management Server (PMS). At registration time, the Application Server and broker negotiate Service Level Agreements (SLAs) containing capability information such as maximum response time, minimum availability or fixed price. These SLAs are stored at the Service Directory Repository. The Pool Management Server (PMS) gathers the QoS characteristics from the monitors in its pool. At regular time intervals, the PMS pushes this information to the Pool Management Server Coordinator (PMSC). This way the PMSC has an overview of the QoS of the total platform. When selecting the services, the FPS can also request the PMSC more detailed QoS information about specific pools or servers. In order to alleviate the users from privacy and security issues, an additional component is present in the framework, namely the Credential Manager. Users only have to authenticate to the framework (by means of eID, smart card or login/password). Once authenticated, the Broker Facade will investigate the service requests, query the needed credentials from the Credential Manager and forward the credited requests to the MSF.

E. Privacy and security

As already stated, the Credential Manager alleviates users from privacy and security issues. Relationships and profiles for access control, when using multiple services, cannot be stored by each individual service. Therefore the Credential Manager can be seen as an Assertion/Credential Federation, grouping all these assertions. By adjusting requests with the required credentials, the brokering middleware supports "content and role based" access control and filtering. Services then only have to map the profiles onto authorization access rights. By only loading users with authentication, the brokering middleware is open for all kind of users, both technical and non-technical.

F. Scalability considerations

In the middleware platform scalability is ensured both in the broker domain as well as in the provider domains.

Scalability in the broker domain is ensured by avoiding single point-of-failures and bottlenecks by replicating components. The architecture contains one or more linked Service Directory Repositories, containing all services offered by the Application Servers described in detail, as well as the Service Level Agreements (SLAs). Since all Service Directory Repositories are referencing each other, it appears like one single SDR and scalability is ensured. Besides the SDR, multiple instances of the other components can exist as well, clustered and interacting with the system simultaneously.

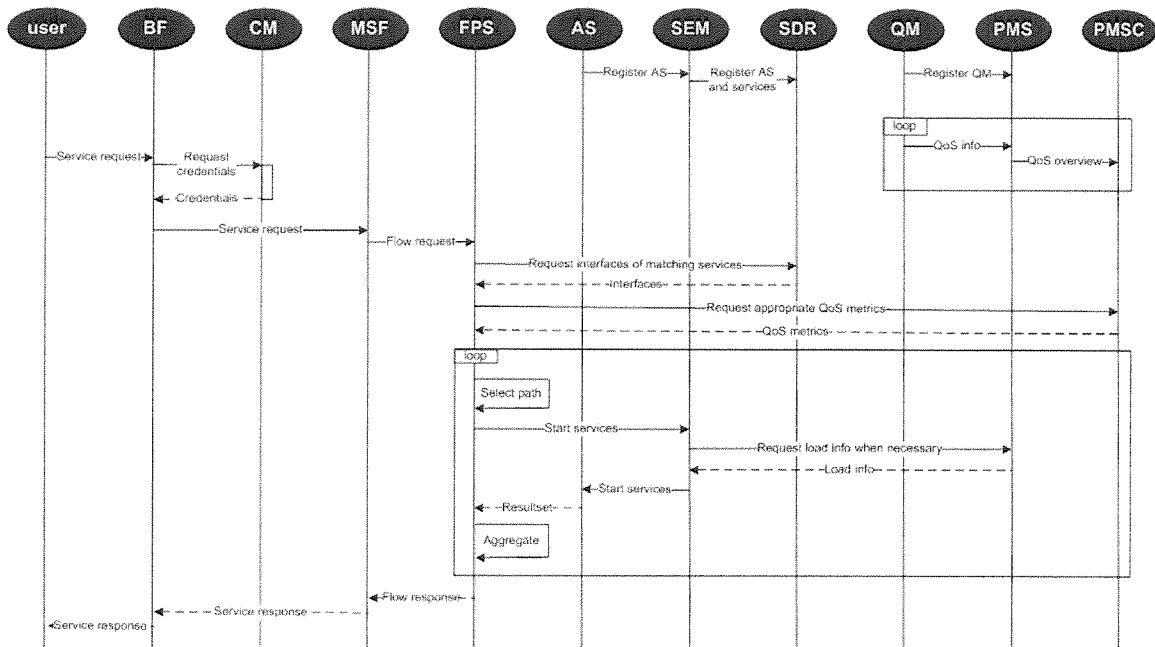This architecture ensures also scalability in the provider

Fig. 4. UML sequence diagram of QoS brokering middleware. The communication between the following components is shown: Broker Facade (BF), Credential Manager (CM), Mapping Service Flow (MSF), Flow Processing Server (FPS), Application Server (AS), Service Execution Manager (SEM), Service Directory Repository (SDR), QoS Monitor (QM), Pool Management Server (PMS) and Pool Management Server Coordinator (PMSC).

domains by organizing the Application Servers into pools. Pools can contain multiple servers, each belonging to one single provider. A provider however can choose to create multiple pools, for example grouped by functionality or performance.

III. BROKER PERFORMANCE

Our broker currently uses Oracle's BPEL engine for the BPEL4WS flow processing and Oracle9i database as dehydration store. Since Oracles BPEL engine is a native BPEL process manager, this result in both performance and scalability advantages over other solutions.

However for having an idea of the performance overhead introduced by using the broker, we did some performance tests. At first, in order to know the invocation times of the BPEL engine, we tested the overhead of calling a single Web service through the BPEL engine, versus calling the Web service directly. The results can be found in figure 5. As can be seen in this figure, calling the Web service via the BPEL engine, generates an overhead of 10 a 20 percent, compared to calling the Web service directly. The overhead, or thus difference between these two time curves, is the invocation time of the BPEL engine, in function of the number of simultaneous requests.

However implementing the service functionality into BPEL itself as a BPEL process, without calling a standalone Web service (neither directly or through BPEL), results in huge performance gains. Unfortunately, besides the huge performance

gains, implementing service functionality into BPEL itself as a BPEL process is limited to simple data manipulation using the basic and structured BPEL Activities and XPath expressions. In order to build more sophisticated applications, either simple or composed services, one has to call external Web services through the BPEL engine. There are however BPEL extensions like BPELJ [12] that allow extra functionality in BPEL processes.
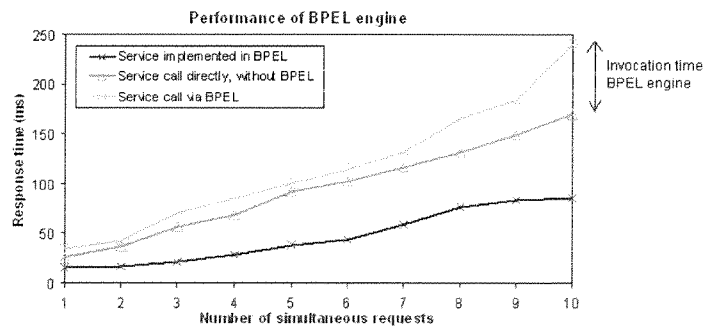
Fig. 5. Response times of calling Java Web service directly compared to calling through BPEL engine

BPEL is however a language for composing multiple services into an end-to-end process. Now we know the engine's performance, we can investigate the overhead of dynamically selecting the services by the broker. Figure 6 presents the

performance differences for composing two Web services using static endpoints versus dynamically selected endpoints in function of the number of simultaneous requests. As can be seen in the figure, choosing at runtime between multiple equivalent services for fulfilling QoS requirements of the users, results in a performance loss. A Web service in the composition can however become unreachable by a number of circumstances, which requires run-time actions to replace it by another compatible Web service. A Web service can also perform less well than guaranteed, which also requires action in order to fulfill QoS requirements. By using dynamic endpoint selection, the broker can act at run-time in order to fulfill QoS guarantees. The broker middleware will also hide for the users if it concerns a simple or composed service and will thus simplify required user interactions. These advantages by using the broker middleware will cost some performance, but since composition standards like BPEL4WS and OWL-S are still developing and tools are not yet optimized to provide this technology, one can still expect optimizations in performance.
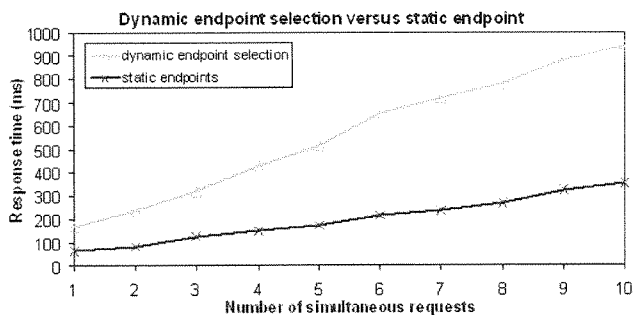
Fig. 6. Response times for composing two services using dynamic endpoint selection versus static endpoints

IV. CONCLUSIONS AND FUTURE WORK

In this paper we presented a Web service based brokering middleware platform, offering dynamic selection, composition and automatic load balancing of services, guaranteeing QoS requirements. Those Application Servers and Web services may be implemented by different vendors, using different programming languages and data definitions. Service providers only need to provide a Web service interface on their applications and the platform takes care of all the rest.

The middleware platform is built around a Broker Facade which intercepts all incoming service requests and makes sure those requests are forwarded to the best Application Server, even for composed services. The Broker Facade forwards these service requests to the Mapping Service Flow component which translates the service requests into an internal composition flow chart. This Broker Facade abstracts the underlying technical middleware and simplifies required user interactions.

Besides simplifying required user interactions and service provisioning, the middleware platform is also highly scalable at both broker and provider level.

These advantages, obtained by using the broker middleware, will cost some performance, but since composition standards like BPEL4WS and OWL-S are still developing and tools are not yet optimized to provide this technology, one can still expect optimizations in performance.

Due to the generic approach, this brokering middleware covers a wide range of application cases in Multimedia Content Delivery, as well as in eHealth and B2B.

We will continue the design of advanced broker algorithms for selecting and composing the services, fulfilling QoS requirements. We will also extend the FPS component in order to support OWL-S in future as well. Moreover, single point-of-failure components will be splitted into multiple instances that run simultaneously in order to increase the scalability of this platform.

V. ACKNOWLEDGMENT

REFERENCES

[1] S. Van Hoecke, W. Haerick, G. De Jans, F. De Turck, E. Laermans, B. Dhoedt, P. Demeester, Design and Implementation of a Secure Media Content Delivery Broker Architecture, The 2005 International Symposium on Web Services and Applications (ISWS'05), Las Vegas, USA, 2005.
[2] A Darwin Partners and ZapThink Insight, Using Web Services for Integration, http://www.xml.org/xml/wsi.pdf, 2002.
[3] T. Andrews, F. Cubera, H. Dolakia, J. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, S. Weeravarana, Business Process Execution Language for Web Services, 2003.
[4] The Business Process Modeling Language, Business Process Management Initiative, http://www.bpmi.org/.
[5] Web Service Choreography Interface, W3C, http://www.w3.org/TR/wsci/.
[6] The OWL Services Coalition, OWL-S: Semantic Markup for Web Services, Technical White paper (OWL-S version 1.1), 2004.
[7] A. Ankolekar, M. Burstein, J. R. Hobbs, O. Lassila, D. L. Martin, S. A. McIlraith, S. Narayanan, M. Paolucci, T. Payne, K. Sycara, H. Zeng, DAML-S: Semantic Markup For Web Services, Semantic Web Working Symposium, California, 2001.
[8] K. Nahrstedt, J.M. Smith, The QoS Broker, IEEE Multimedia Magazine 2(1), 1995.
[9] T. Yu, K. Lin, The Design of QoS Broker Algorithms for QoS-Capable Web Services, Proceedings of the 2004 IEEE International Conference on e-Technology, e-Commerce and e-Service, 2004.
[10] A. Barak, O. La'adan, The MOSIX Multicomputer Operating System for High Performance Cluster Computing, Journal of Future Generation Computer Systems, 13 (4-5) (1998) 361-372.
[11] Unicore, http://www.unicore.org.
[12] IBM and BEA, BPELJ: BPEL for Java technology, http://www-128.ibm.com/developerworks/library/specification/ws-bpelj/

SWWS'06

The 2006 International Conference on Semantic Web & Web Services

Foreword   Author's Index

Session: WEB SERVICES

The 2006 World Congress in Computer Science, Computer Engineering, and Applied Computing

Monte Carlo Resort, Las Vegas, Nevada, USA
June 26–29, 2006

Conferences:

The 2006 International Conference on Bioinformatics & Computational Biology

The 2006 International Conference on Computer Design & International Conference on Computing in Nanotechnology

The 2006 International Conference on Computer Graphics & Virtual Reality

The 2006 International Conference on Communications in Computing

The 2006 International Conference on Scientific Computing

The 2006 International Conference on Data Mining

The 2006 International Conference on e-Learning, e-Business, Enterprise Information Systems, e-Government, & Outsourcing

The 2006 International Conference on Engineering of Reconfigurable Systems & Algorithms

The 2006 International Conference on Embedded Systems & Applications

The 2006 International Conference on Foundations of Computer Science

The 2006 International Conference on Frontiers in Education: Computer Science & Computer Engineering

The 2006 International Conference on Grid Computing & Applications

The 2006 International Conference on Artificial Intelligence

The 2006 International Conference on Internet Computing & International Conference on Computer Games Development

The 2006 International Conference on Wireless Networks

The 2006 International Conference on Information & Knowledge Engineering

The 2006 International Conference on Image Processing, Computer Vision, & Pattern Recognition

The 2006 International Conference on Machine Learning; Models, Technologies & Applications

The 2006 International Conference on Modeling, Simulation & Visualization Methods

The 2006 International Conference on Parallel & Distributed Processing Techniques & Applications & International Conference on Real-Time Computing Systems & Applications

The 2006 International Conference on Pervasive Systems & Computing

The 2006 International Conference on Security & Management

The 2006 International Conference on Software Engineering Research & Practice & International Conference on Programming Languages and Compilers

The 2006 International Conference on Semantic Web & Web Services

WORLDCOMP'06

June 26 - 29, 2006

Las Vegas, Nevada, USA

PDPTA/RTCOMP'06
SERP/PLC'06
BIOCOMP'06
IPCV'06
MSV'06
CGVR'06
ERSA'06
CDES/CNAN'06
ESA'06
ICOMP/CGD'06
SWWS'06
ICAI'06

MLMTA'06
SAM'06
DMIN'06
IKE'06
CSC'06
GCA'06
ICWN'06
PSC'06
CIC'06
FECS'06
EEE'06
FCS'06

Editor: H. R. Arabnia
University of Georgia, GA, USA