SDPS-2012
Printed in the United States of America, June, 2012

SERVICE DESIGN STUDIO (SDS) - THE EXECUTION ENVIRONMENT FOR THE SERVICE DESIGN STUDIO

Hülya Iscan
Fraunhofer ISST, Emil-Figge-Straße 91, 44227 Dortmund, Germany
huelya.iscan@isst.fraunhofer.de

Stephan Flake, Jürgen Tacken
Orga Systems GmbH, Am Hoppenhof 33, 33104 Paderborn, Germany
{sflake,jtacken}@orga-systems.com

Martin Ley
tarent solutions GmbH, Rochusstraße 2-4, 53123 Bonn, Germany
m.ley@tarent.de

Christian Schmülling
Talend GmbH, Servatiusstraße 53, 53175 Bonn, Germany
cschmuelling@talend.com

ABSTRACT

Cloud computing, virtualization and Service-oriented Architecture (SOA) enable a synergistic combination of IT and logistics services to use flexible, cost-saving and innovative logistics solutions especially for small and medium-sized enterprises (SMEs). The joint project Service Design Studio (SDS) is a research project of the Leading-Edge Cluster 'EffizienzCluster LogistikRuhr', contributing to the topic 'Logistics-as-a-Service'. The project has the primary goal to develop a Web-based, cloud-oriented service platform which allows utilizing existing logistics IT services as secure and billable commercial products via cloud solutions and/or software-as-a-service platforms like, e.g., Amazon Web Services or the Logistics Mall (Fraunhofer Innovation Cluster, 2012). In this article, we present a prototypical execution environment for the SDS as the result of a Proof-of-Concept implementation to demonstrate the feasibility of the approach.

KEYWORDS

Cloud Computing, SOA, Service-oriented market places, Logistics, Security, Billing, USDL, Integration, XaaS

INTRODUCTION

The logistic industry has a turnover of more than 200 billion € per year only in Germany and is the third largest industry following the automotive and engineering sector. A lot of logistics companies do not have sufficient IT expertise, capacity or capital required to close the gap between requirements and status quo in logistics IT. Therefore, logistics companies need flexible, favorable and short-term solutions to support their logistics processes. One general approach for this is to deploy software over the Internet (Meinhardt and Lippmann, 2010). This means no more traditional desktop applications are provided but specific IT services, e.g., a route planner. In cloud computing, applications and data of a company no longer reside on local computers or in a corporate data center, but 'in the cloud' as a metaphor for virtual, elastic resources accessible via the Internet (Mell and Grance, 2011).

The flexibility and dynamic of logistics IT processes has now reached a level that can only be handled on the basis of efficient logistics IT solutions. This requires also new methods and tools for accounting and security for such logistics IT services (Dillon, et al., 2010). Investment in one's own infrastructure and the corresponding know-how can largely be avoided. A needs-based access to rented customized software solutions via a standard Web browser becomes possible. Billing is done based on usage of resources; the user does not pay for unused features. This is in contrast to the expensive, monolithic, multi-featured logistics IT solutions, e.g., Warehouse Management Systems, available today.

The basic idea here is to support logistics IT services on demand. Customers benefit from cost and performance transparency by the pay-per-use- and XaaS-principle of the cloud. This creates the opportunity to acquire a wide range of services from a single platform. Users as well as providers benefit from such an infrastructure, in which a multitude of cloud-based logistics-related services and

software can be offered and even orchestrated by means of business processes.

Once such virtual service-oriented marketplaces for the domain of logistics IT services have been established, it should be possible to search and compare these services not only according to their functional properties, but also according to non-functional properties such as service level agreements (SLAs) and cost models (Cardoso, et al., 2010). However, for an automated semantic search and comparison, the description of the required aspects has to be interpretable by a 'trading service', which matches the required aspects against the services' semantic description of functional and non-functional aspects in the given execution environment. Consequently, existing logistics IT services have to be extended using a dedicated tool, such that they can be deployed and offered in a SOA and/or cloud environment (Papazoglou, et al., 2007). To reach these multi-objective goals, the SDS project has combined different standards for the definition of a holistic semantic service description that encompasses (a) a functional and business view, (b) business object descriptions in a domain and technical view, and (c) process model descriptions on different abstraction levels. The result presented in this article is a first design environment to make existing logistics IT services available in different cloud platforms (e.g., Amazon, Google, or the Logistics Mall (Fraunhofer Innovation Cluster, 2012)). Whereas the requirements and state-of-the-art have already been described in (Steinbuß, et al., 2012), this article focuses on the design and implementation of the tools and services for an end-to-end Proof-of-Concept (PoC) implementation.

Please note that it is still open who is going to commercially provide such a design environment. In the first place, it could be the operator of a virtual marketplace aiming at lowering the entry barrier to service providers, who in turn would like to offer their existing IT services as Software-as-a-Service. But maybe a completely new role will come up, offering a 'cloudification' service independently from a single, dedicated target platform.

PROBLEM FORMULATION

The main objective of the SDS project is to develop and provide methods and tools to enable the deployment of existing logistics IT services in cloud and SOA environments based on the Software-as-a-Service (SaaS) model. An end-to-end PoC implementation shall demonstrate the feasibility of the approach. The PoC shall encompass a tool (called the SDS Tool in the remainder) for the specification of functional and non-functional aspects and the export of a corresponding, standard-based semantic service description to a sample target cloud platform.

APPROACH

The starting point for working with the SDS Tool is an existing logistics IT service given with a functional service description (e.g., a WSDL). With the SDS Tool, non-functional properties, in the following called aspects, can be specified, like the costs of service usage, service level agreements and security aspects like authentication and authorization. After the specification, the existing functional service description will be enhanced with these aspects. For this enhancement the upcoming W3C standard 'Universal Service Description Language' (USDL) (W3C, 2011) will be used, which has been developed in the Theseus TEXO[1] and other research projects (SAP, 2011). The SDS Tool therefore features existing logistics IT services with a functional service description to refine and wrap domain-specific packages around them (cf. Figure 1).

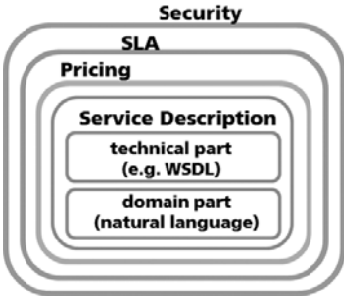

Fig. 1 SDS packaging

The description of the packages is a semantic service description (SSD) based on USDL. It describes on the one hand what the service is and how it is technically represented and on the other hand the non-functional aspects for either the complete service or even for single service operations. With the SSD, single elements can be described and then linked together (cf. Figure 2).

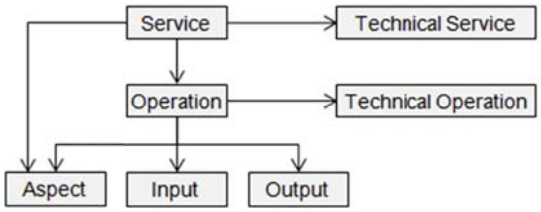

Fig. 2 Semantic service description model

A business service ('Service' in Figure 2) is typically linked with a technical service and all business operations ('Operation' in Figure 2) are linked with technical operations. In the current version, we use Web services as

[1] http://theseus-programm.de/de/texo.php

technical representations. However, the concept is open for other techniques, too. An aspect describes a non-functional property. Within the SSD, we are able to link business services and business operations with aspects. That means one aspect is linked to the whole service or to at least one operation of the service.

An aspect is a generic construct. Concrete aspects could be security aspects, like authentication and authorization aspects, or billing aspects. Within these aspects, it is possible to add configuration information. For example, it is possible to specify roles which are allowed to access a service or which price plan should be applied when executing the service.

SSD is a smart solution specially tailored to our needs. It currently is a proprietary format. However, it can be transformed from and to USDL**.** USDL is a generic, extensive approach for describing all relevant elements in a service-oriented world. An overview of the different USDL modules is depicted in Fig. 3. USDL is currently being evaluated by the W3C to become a new standard (W3C, 2011).

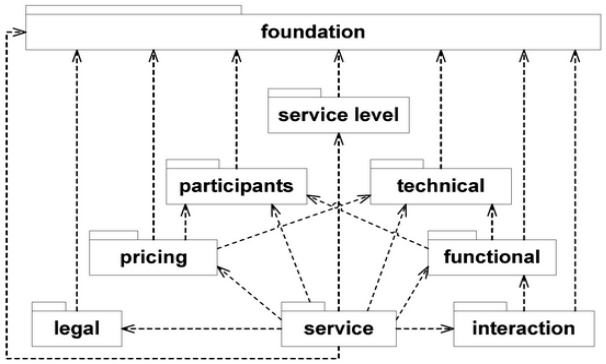

Fig. 3 Modules of USDL (SAP AG, 2011)

For the SSD in the SDS project, we just need a subset of the USDL capabilities: For the security aspects in SSD, the Service Level Agreements (SLA) specifications available in USDL can be used. For the billing aspects, we can use the pricing module in USDL.

But there is an important feature missing in USDL. This is the link between SLAs or price plans and operations. To overcome this, we used expressions with our own expression language to link the corresponding USDL elements for SLAs and price plans to operations. Consequently, we are able to transfer the SSD completely to USDL.

In the following two subsections, details about the implementation of the security and billing aspects will be provided.

Security Aspect

It is planned to deploy the services enhanced by the SDS Tool in a cloud environment, where we are facing some additional security concerns. In the cloud, the different components are not kept in firewalled zones as known from traditional hosting, but may even be distributed in different data centers. This requires a protection of the communication between the components. Also, we should take the security promises of the cloud providers with a grain of salt, because we cannot review the inner processing mechanisms of the cloud infrastructure. Furthermore, we have to keep in mind that our deployed software is not operating on real hardware, but in virtual servers or runtime environments, which can be completely observed by the cloud operator. If an attacker finds and exploits vulnerability in the hypervisor of the cloud system, it is possible to break out of the virtual server domain and gain control over the cloud infrastructure. We are also dealing with a multi-tenant environment, where many clients are using the same server or database, requiring a strict isolation between the customers, such that no data is unintentionally being shared between them. To prevent most of these attacks, it is recommended to encrypt all stored data and – in an Infrastructure-as-a-Service (IaaS) environment – also the virtual service disk images (Varia, 2011).

Billing Aspect

For rating, charging and billing of the usage of services, an Online Charging System (OCS) is utilized to determine usage fees and perform monetization-based decisions (e.g., threshold notifications) in real-time. The SDS OCS holds all monetary and non-monetary information relevant for rating and charging about services and their users. Users have several balances for different purposes filled with monetary or non-monetary units. Thus, a ‘price’ may be expressed in monetary units, but can also refer to non-monetary units like reward points, bonuses etc. Also the revenue sharing among SDS operator, mall operators, and cloud operators is performed in the SDS OCS. In addition, the SDS OCS can initiate blocking of service usage, e.g., when the credit-worthiness of a customer is not clear.

Rating, charging and billing in the context of the SDS execution environment can be realized in different ways. On the one hand, an SDS operator can offer mall operators (i.e., the operator of a cloud-oriented virtual marketplace) all of these functionalities as a managed service. On the other hand, it is also possible that a mall operator is running an own billing system. The challenge here is therefore to come up with appropriate platform-independent, standardized interfaces that cater for all potential deployment and access options for rating, charging, billing and related user management functions. The approach in the SDS project is to define generic interfaces for the SDS OCS as described below in more detail. Third-party billing software that is running at different mall operators and following these interfaces can then be easily integrated with the SDS execution

environment. However, note that the SDS operator can in addition still use an own SDS OCS for control purposes, e.g., for revenue assurance.

Figure 4 shows the architecture of the SDS OCS. The core of this subsystem is an online charging system that is able to support real-time rating, charging and billing. In the SDS project, Orga Systems' OPSC Gold convergent charging and billing system is used. Of course, other OCSs could also be employed instead.

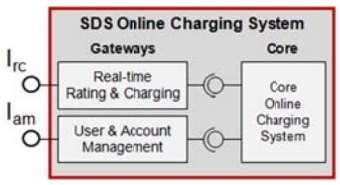

Fig. 4 Architecture of the SDS OCS

Other system elements can communicate with the SDS OCS only via two gateways. One gateway is responsible for real-time rating and charging operations (l_{rc}) and the other one for user and account management operations (l_{am}).

The User and Account Management component supports the registration of users (or: customers) to the OCS and the management of accounts, e.g., retrieving and updating current balances and obtaining transaction histories.

The Real-time Rating & Charging component allows to perform rating and charging either directly or by means of reservations in a session-based manner (i.e., by subsequent reserve, extend, and charge operation calls).

For the gateways, the SDS OCS is building upon existing Web Service standards as defined by the GSMA OneAPI[2]. The OneAPI initiative defines a commonly supported set of lightweight and Web-friendly APIs to allow network operators to expose useful network information and capabilities.

The interfaces for the user and account management as well as the real-time rating and charging gateway are explained in more detail in the following paragraphs. Note, however, that the functionalities for User and Account Management as well as for Rating are not covered by the OneAPI standard yet. Nevertheless, in our implementation, we always followed the philosophy of RESTful Web Services and the terminology of the OneAPI.

User Management. When registering a customer, some information about the customer needs to be provided. Currently the following information in the SDS OCS:

- A unique customer number,
- a role (one of CloudOperator, MallOperator, SDSOperator, ServiceProvider, EndUser),
- customer data (such as first name, last name, country, zip code, city, street, e-mail, etc.).

The operations provided by this interface are:

- registerUser: This operation registers a new customer in the SDS OCS.
- changeRegistration: This operation allows to change the information about a customer.
- deRegisterUser: This operation invalidates the registration of a customer.
- getSubscription: This operation allows to retrieve the subscription data of a customer to a service.
- subscribe: This operation allows to set the subscription of a user to a service.
- unsubscribe: This operation allows to remove the subscription of a user to a service.

Account Management. For the account management, operations supporting the retrieval and manipulation of balances and transaction histories are provided. A balance is characterized by

- a balance ID, e.g., 'Main Money' or 'Free Services',
- a balance value, which is the current value of a balance as a floating point number,
- a unit value, identifying the monetary or non-monetary balance unit, e.g., 'EUR', 'USD', 'Minutes', or 'CreditPoints'.

The operations provided by this interface are:

- getBalance: This operation allows to retrieve one or more balances of a customer from the SDS OCS.
- updateBalance: This operation allows to update a balance of a customer in the SDS OCS.
- setBalance: This operation allows to set a balance of a customer in the SDS OCS.
- getHistory: This operation allows to retrieve a history of transactions of a customer's account.

Rating and Charging. For rating, an interface that supports the retrieval of the current price for a service is provided. The operation provided by this interface is:

- rate: This operation allows to retrieve the current price for a service usage.

For charging operations, an interface that supports either direct charging or charging with reservations in real-time is provided. Exactly adhering to the OneAPI for Payment, the operations provided by this interface are[3]:

- amount: This is a direct charging request, where the value for a service will directly be charged. By the same operation, refunding of a previous charged amount can be performed (by supplying the reference id of the charging request to refund).
- amountReservation: Reservation-based charging allows session-based charging with the following actions:

[2] http://oneapi.gsma.com

[3] http://oneapi.gsma.com/payment-restful-api-v2-1/

1. Reserve an amount: At the beginning of a session, a certain amount is reserved.
2. Extend the reservation: If necessary, a reservation can be extended by an additional amount.
3. Charge a reservation: At the end of a session, the reservation can be charged.
4. Release a reservation: If necessary, a reservation can be released.

EXECUTION ENVIRONMENT

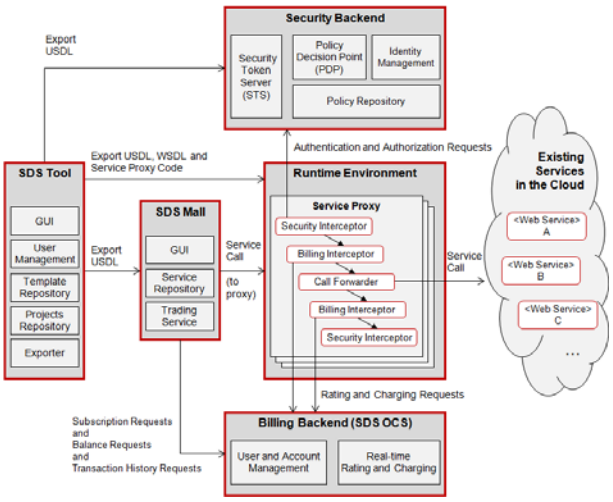

Fig. 5 PoC execution environment

The complete PoC execution environment consists of several components (cf. Figure 5):

- SDS Tool: The SDS Tool is a Web application which consists of a GUI, a user management component, repositories and an exporter component. After creating an SSD and a USDL description, the service can be exported from this environment and deployed to the SDS Mall and to a runtime environment. Furthermore, security-related information has to be provided to dedicated security backend components. The exporter component takes an existing service description from a WSDL file and adds the specified aspects as 'domain capsules' to this service description to build an SSD. After this, the exporter sends the SSD as an XML document to the SDS Mall and the runtime environment.
- SDS Mall: A GUI is utilized for displaying and booking the available services. The trading service supports the search for specific services and considers for each service the associated security aspects and the information for billing.
- Runtime Environment: The runtime environment is the component that supervises the operation calls to the offered services. This is done with a proxy mechanism: Each 'service proxy' represents an executable service and is implemented as a forwarder of incoming requests to the actual service residing in a cloud environment. The proxy is enhanced with 'interceptors' to cater for the additional non-functional aspects.
- Security backend: One challenge is given by the demand of security and the required isolation of all customers in multi-tenant environments, so we implemented a policy-based security infrastructure with authentication and authorization mechanisms.
- Billing backend: For rating and charging the usage of services, an Online Charging System (OCS) is utilized to determine usage fees and perform monetization-based decisions (e.g., threshold notifications) in real-time.

In the remainder, more details about (a) the export process and implementation, (b) the policy-based security infrastructure, and (c) the SDS Mall are provided.

Exporter Implementation and Proxy Generation

The export process is depicted in Figure 6: After an SSD has been created for a service with the SDS Tool the export process can be initiated. One of the main responsibilities of the exporter is to create an enhanced service description in USDL with regard to the service design made in the SDS Tool and send this description to dedicated target platforms, where the service then can be offered to customers. In the PoC, the SDS Mall is the only available target platform.

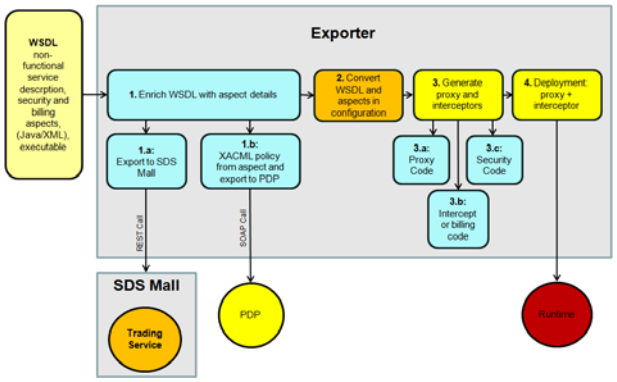

Fig. 6 Export process

For each of the already existing services, a proxy service will be created. The USDL generated by the exporter contains the original service description enhanced with the description of the selected aspects. This information is then used to create the configuration for the proxy services. The last step of the export process is the deployment of the proxy services. The exporter is able to deploy the proxy services automatically with no further interaction.

Besides calling the different interceptors to ensure the consideration of the non-functional aspects like a service proxy has no further business logic. It is just forwarding the request to the original, already running service. When using the proxy all specified aspects are enforced, but it is of course still possible to access the original service without using the proxy service. If we want to make sure that the service can only be used via the proxy service, some kinds of network restrictions are needed.

The implementation of the exporter is based on Apache Maven[4]. Maven is a powerful build tool with many features we can use. We have built a new Maven plugin that enables Maven to work with USDL documents.

The export process is started from the SDS Tool. Within the SDS Tool the SSD is created and stored by the exporter as an USDL document. After that the exporter calls Maven. Technically this means that there is an embedded Maven call from the SDS Tool application to Maven. As input, Maven takes the generated USDL document. As a first step in the exporter process the USDL document is parsed and the SSD representation is generated from it.

Inside the USDL description a WSDL description of the original service is referred. This is the target service to be called by the proxy service. The proxy service itself is based on Apache Camel[5]. Camel is a powerful enterprise-ready integration framework. We need just a very small subset of the capabilities from Camel. Camel is, e.g., able to create complex routes. In the case of the proxy services, dedicated routes are needed from the proxy interface to the target service. For the Camel framework it doesn't matter what the message content is. Within the route configuration, also the Camel components used in the route are configured.

For the final proxy service, Apache CXF[6] is used. CXF is a service framework and is widely used for creating Web Services and Web Services clients. CXF has a feature called 'interceptor', which is used to enforce the specified aspects. With interceptors it is possible to hook in the message flow and even control it. For example, an interceptor may count the service calls and call the SDS OCS once for every call. Or in case of some more detailed billing rules, an interceptor could send detailed information about the calls to the SDS OCS and depending on the answer from the SDS OCS, it is possible to interrupt the message flow. Exactly these features will be used to enforce the specified aspects at runtime.

The export process checks if authentication is specified in the USDL description. If yes, it will generate authentication policies for the proxy WSDL. Policies follow the WS-Policy standards (W3C, 2007). For authentication, there are standard security policies available. For the authorization and billing aspects, we build our own policies and attach them to the WSDL description. Basically, if an aspect is needed, we represent it technically as a policy. CXF is interpreting these policies and tries to handle them. For authentication, it is possible to use standard CXF mechanisms to authenticate a request. CXF adds some security interceptors in the message flow and executes it. We do exactly the same for our own policies for authorization and billing. We register some handlers for the policies. If CXF finds a policy, it will automatically add the necessary authorization and/or billing interceptors to the message flow.

The aspects specified in the USDL description are configurable. For example, it can be configured, which roles are allowed to access a service or whether the service should be charged with a flat rate or per usage. These configurations need to be published to our security backend and/or billing backend. Therefore the exporter is able to register this configuration automatically within the export process. It is calling the different backends and sends them either just the concrete aspect configuration or the complete USDL description.

Security Implementation

The implementation of the authentication is backed by security tokens implemented with the respective well-established Web Service standards (here: WS-Security, WS-Trust, WS-MEX, WS-Policy, WS-SecurityPolicy and SAML with WS-Trust binding). If a service has authentication enabled in the SSD, this will also be reflected in the WSDL of the generated proxy service. If a client connects to the service proxy, it first checks the policy, containing information about the required token type and the endpoint of the Security Token Server[7] (STS). The client negotiates the authentication method with the STS via WS-MetadataExchange. Then the client authenticates itself with the required method against the STS. If the authentication data is valid, the STS issues a SAMLv2 token including the username and sends it back to the client. The client then sends the token to the service proxy.

For authorization, we use a role-based access control (RBAC). For each service operation a set of roles allowed to access it can be defined. Users can access an operation if they are a member of a role which is bound to the operation. Technically the three parts service, operation and role are considered as a privilege. These privileges are transformed to XACML policies and then stored in a policy repository of the policy decision point (PDP). The generated proxy contains a security interceptor acting as

[4] http://maven.apache.org

[5] http://camel.apache.org

[6] http://cxf.apache.org

[7] https://evolvis.org/plugins/mediawiki/wiki/siam/index.php/Security_Token_Service

the policy enforcing point (PEP). The PEP sends an XACML request to the PDP. The PDP generates from the XACML request, the XACML policies in the policy repository an authorization decision. An identity management system acting as a policy information point (PIP) supports the PDP in resolving attributes unknown to the PDP, for example which user is member in which roles.

For creating a deployable file we use the capabilities from Maven to create a Tomcat war-file. Maven then assembles all dependencies directly together. Furthermore, we use the Maven feature to deploy our war-file directly to a Tomcat server. We use the Tomcat Maven plugin[8] for this. With the deployment, the export process is completed. The Tomcat server is automatically starting the proxy (Camel with CXF endpoints) and waiting for requests.

SDS Mall and Trading Service Implementation

The SDS Mall is our prototype implementation of a virtual multi-provider marketplace operated by a so-called 'Mall Operator'. From the point of view of the SDS, this is a possible target platform to which IT services with SSDs can be exported via Web Service calls. The lifecycle management operations of the corresponding service import interface are:

- addServiceOffer: This operation allows to install a new service offer into the SDS Mall.
- changeServiceOffer: This operation allows to change an existing service offer in the SDS Mall, e.g., by adding a new price plan .
- deactivateServiceOffer: This operation allows to deactivate an existing service offer in the SDS Mall. Note that existing active subscriptions are not affected.

Through its graphical user interface, the SDS Mall is offering the following views to end users:

- 'Account': This view provides an overview on the account data with the current balance and a configurable number of the last transactions performed.
- 'Book Services': This view provides an overview of all available services. We are currently developing an intelligent search ('Trading Service') that allows users to apply additional filters to the list of available services. In particular, it should be possible to filter the search results against information provided in the SSD, e.g., security, billing, and SLAs (e.g., a search can be restricted to show only the services that are cheaper than a specified maximum amount per period – note that a specific logic has to be applied here to be able to compare arbitrary price plans to the specified filter).
- 'Use Services': This view provides an overview of all current subscriptions together with direct links to access the corresponding services via Internet.
- 'Options': In this view, language and layout settings can be made.

Figure 7 shows a screenshot of the view 'Book Services' with an already selected service offer and a corresponding list of available tariffs (or: price plans). All service information shown here (security info, tariff descriptions) has automatically been extracted from the SSD exported by the SDS Tool.

Fig. 7 SDS Mall screenshot

Service Call

The process of calling a service starts at the Mall side. The Mall will call the proxy service generated by the exporter process. In addition to the information needed by the original service, the call needs to carry the information necessary for the specified non-functional aspects. If authentication is required, the Mall needs to be able to authenticate against a secure token service (STS). In the PoC, we are using CXF as consumer technology for creating the proxy service. CXF owns this feature directly. So we just need to configure the call from the Mall to be able to send authentication information to the STS. On

[8] http://tomcat.apache.org/maven-plugin.html

generating the proxy service, the CXF framework interprets the service WSDL with the added policies. It will recognize that authentication is needed and will add all the needed interceptors automatically. The STS is automatically called for creating a token. This token is then added to the proxy service call that should be performed by the Mall.

During the service call the request is received on the service side and the token will get validated. Furthermore, the authorization and billing policies will be interpreted. If authorization is needed, an authorization interceptor is called. It will extract the user information from the request and send them to the security backend. The result decides about processing or stopping the message flow. Exactly the same will happen for the billing aspects. Just a different backend, i.e., the SDS OCS, is called. After all interceptors have been successfully executed, the call will be forwarded to the original service.

SUMMARY AND OUTLOOK

This article provided a brief overview of the current state of the SDS project and the first complete deployment of the design tools and PoC execution environment.

The current PoC execution environment is based on Apache CXF and Oracle Metro on Amazon EC2. All aspects are mapped to existing Web Service standards like WS-Trust or WS-Policy. However, the system components are currently distributed and reside on different project partners' servers. This adds additional complexity to the management of the PoC execution environment. We plan to migrate as many components as possible onto a single cloud environment to ease the further developments in the upcoming phases.

Furthermore, we are going to integrate more closely with the Logistics Mall project, which is a highly interesting cloud target environment for the Service Design Studio. Additionally we will compare our results with related works like, e.g. the Software-Cluster[9] and ADIWa[10]. We are also improving the Trading Service with advanced features for searching for services by taking, e.g., security and billing and other SLA aspects into consideration.

For further improvements, user feedback from the other projects contributing to the topic 'Logistics-as-a-Service' is very helpful. For example, the partner project 'Supply Chain Execution' provided new requirements for our billing aspect with respect to revenue sharing.

ACKNOWLEDGMENTS

The project Service Design Studio is funded by the German Federal Ministry of Economics and Technology in the context of the High-Tech Strategy for Germany with the support code 01IC10L23. The Service Design Studio is part of the Leading-Edge Cluster LogistikRuhr (http://www.effizienzcluster.de) in the leading topic 'Logistics-as-a-Service'.

9 http://www.software-cluster.com/en/

10 http://www.adiwa.net/

REFERENCES

Cardoso, J., Barros, A., May, N. and Kylau, U., 2010, "Towards a Unified Service Description Language for the Internet of Services: Requirements and First Developments", *7th International Conference on Services Computing (SCC 2010)*, IEEE Computer Society Press, pp. 602-609.

Dillon, T., Wu, C. and Chang, E., 2010, "Cloud Computing: Issues and Challenges". *24th IEEE International Conference on Advanced Information Networking and Applications (AINA 2010)*, IEEE Computer Society Press, pp. 27-33.

Fraunhofer Innovation Cluster, 2012, "Logistics Mall – Cloud Computing for Logistics". Homepage: http://www.ccl.fraunhofer.de/en (last accessed on March 8, 2012)

Meinhardt, M. and Lippmann, T., 2010, "Cloud Computing für Logistik: Akzeptanz und Nutzungsbereitschaft der Logistics Mall bei Anwendern und Anbietern", Fraunhofer Verlag, ISBN 978-3-8396-0220-1.

Mell, P. and Grance, T., 2011, "The NIST Definition of Cloud Computing", *NIST Special Publication 800-145*. Available at http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf (last accessed on March 8, 2012)

Papazoglou, M. P., Traverso, P., Dustdar, S. and Leymann, F., 2007, "Service-Oriented Computing: State of the Art and Research Challenges", *Computer*, IEEE Computer Society, Vol. 40, No. 11, pp. 38-45.

SAP AG, 2011, "Unified Service Description Language 3.0 (USDL) Overview". Available at http://www.internet-of-services.com/fileadmin/IOS/user_upload/pdf/USDL-3.0-M5-overview.pdf (last accessed on March 8, 2012)

Steinbuß, S., Flake, S., Ley, M., Schmülling, C. and Tacken, J., 2012, "Service Design Studio for SaaS". To be published by Springer in "Efficiency and Logistics".

Varia, J., 2011, "Amazon Web Services – Architecting for the Cloud: Best Practices". Available at http://media.amazonwebservices.com/AWS_Cloud_Best_Practices.pdf/ (last accessed on March 8, 2012)

W3C, 2007, "Web Services Policy 1.5 – Framework". W3C Recommendation 04 September 2007. Available at http://www.w3.org/TR/ws-policy/ (last accessed on March 8, 2012)

W3C, 2011, "W3C Unified Service Description Language Incubator Group". Homepage: http://www.w3.org/-2005/Incubator/usdl/ (last accessed on March 8, 2012)