Claudio Di Sipio

Andrea Bianchi

Lorenzo De Lauretis

CCMS: Custom conference management system

Service oriented software engineering

Summary

[Introduction 2](#_Toc507410123)

[Architecture: Component diagram 2](#_Toc507410124)

[Architecture: Sequence Diagrams 3](#_Toc507410125)

[Describing services: WSDL files 6](#_Toc507410126)

[API usage and project structure 9](#_Toc507410127)

# Introduction

This document describes our application that allows users to handle the participation to scientific conferences in a smart and simple way through the usage of a web application.

For each of them, the user knows the place and the date when the conference takes place. We are assuming that the user stores all the information about the conference on the Google Calendar.

Therefore, by using the provided API Google Calendar and Google Maps he/she can retrieves a list of all possible conferences of interest with the related place (the map). Regarding the map, the application shows to user also restaurant, B&B and other POIs near the place of the conference.

Furthermore, he/she can download in advance a short preview of all presentations that occur in that conference, via a service provider implemented from scratch. Finally, for the sake of completeness, our latter service uses a simple MySql DB that stores the information about conferences.

From the managed data point of view, we will have in the DB information related to conferences, such as an image, programs and previews (a file). The file linked to a preview can contain a short description of a presentation like topic, authors and so on.

In the next sections, we will provide the complete list of the component and the interactions among them in order to provide all requested system functionalities.

# Architecture: Component diagram

Here below there are all components that are our Web services involved in the system and performing all the functionalities:

* Conference Application (Mobile App)
* Conference Manager (Prosumer)
* Google Maps (Provider API)
* Google Calendar (Provider API)
* Preview service (Provider)
* MySQL DB

The architecture of the system and the corresponding required/provided interface are described by the following component diagram. Notice that the two of the providers are inherited and exploit Google API while the provider for the manage conference information will be implement from scratch.

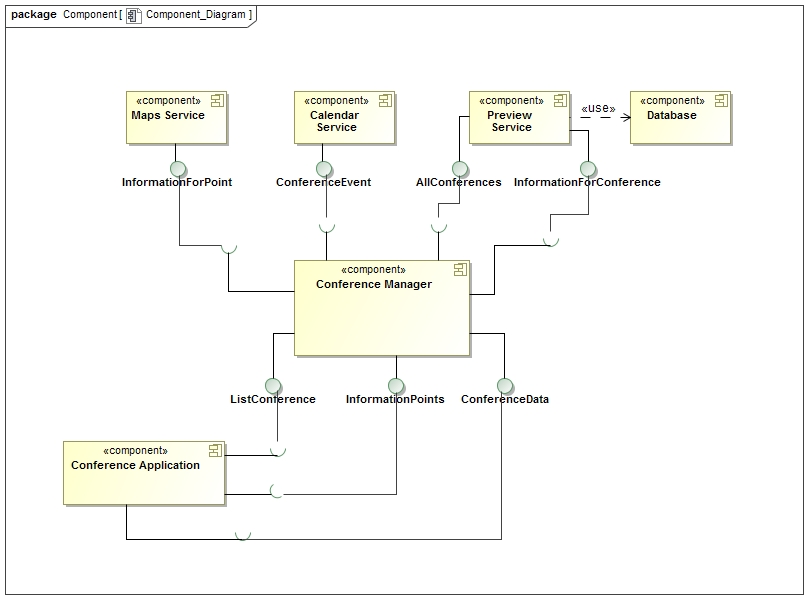


Figure 1: Component Diagram

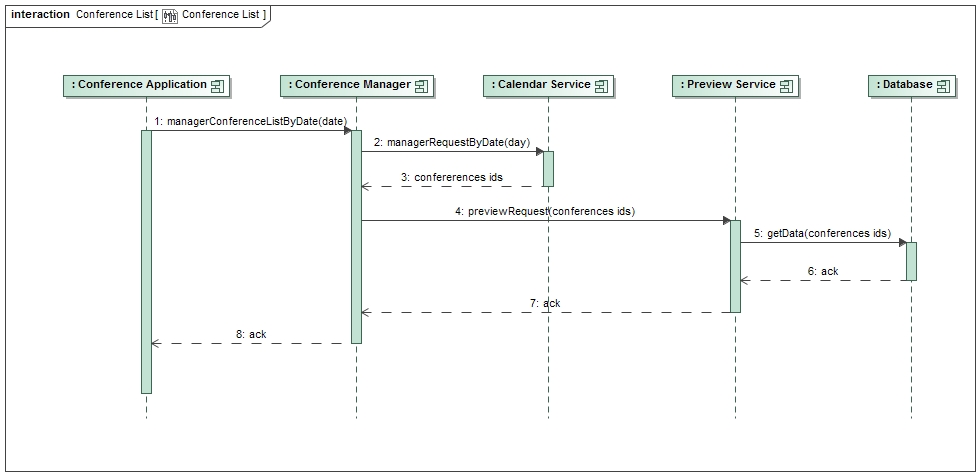
In particular, the *Calendar Service* stores the ID of conference, the name and date that are used to show map and information while the *Preview Service* interacts with the DB and provides all previews.

# Architecture: Sequence Diagrams

Regarding the interactions with the system, the first type of interaction is when the user wants to retrieve the lists of all conferences stored in Google Calendar and also an image from the DB using a numeric ID, using the web application that interacts with the Conference Manager. Notice that the Google Calendar is integrated with the one on the mobileapp that acts as the client.

Another type of interaction is about the retrieval of the previews of a certain conference, that briefly describe the content of a specific conference such as the presentations that will take place, the authors, the order of appearance, the scheduling of the day and so on.

Finally, the last interaction is when the user should be interested in get the position and near POIs of a specific conference. As we can see, this task is managed by the prosumer (Conference manager) that perform the request using Maps provider once get the event from Calendar.

All these possible interactions are described by the sequence diagrams here below.

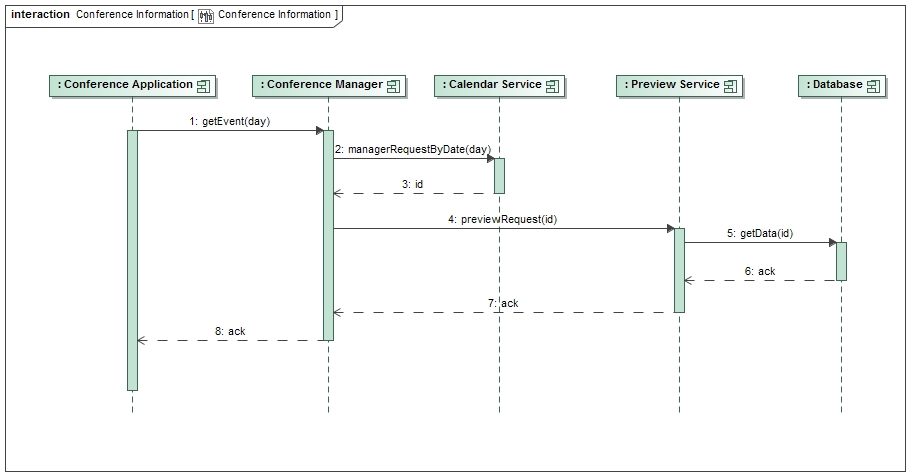


Figure 2: Sequence Diagrams for Previews and List of conferences

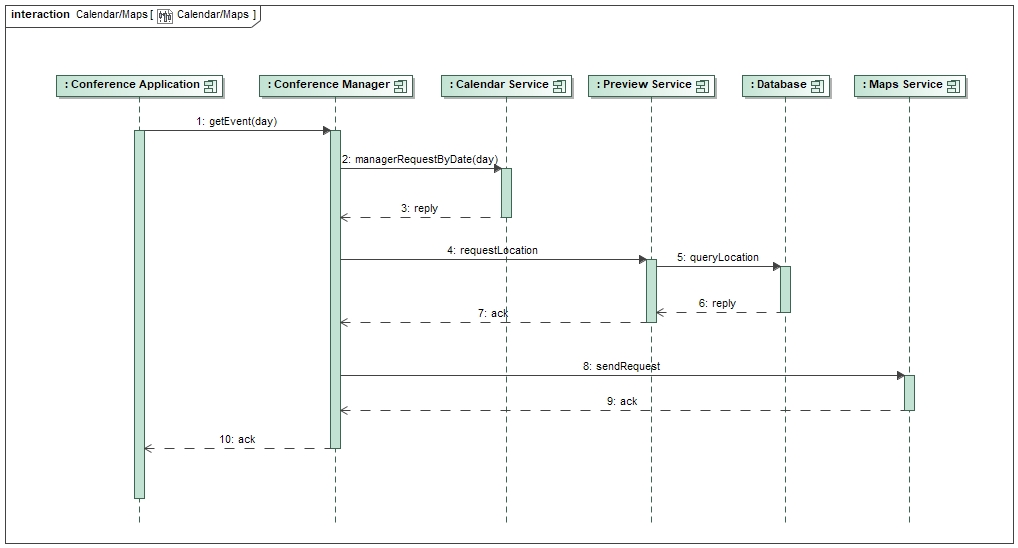


Figure 3: Sequence diagram for Map

Note that, regarding the interactions between the prosumer and the providers, there are two main tasks that the system performs: the prosumer asks to *Calendar Service* to provide the event and request for this the position and only after the response it can refers to *Maps Service* the POIs while the second interaction is related to the preview table; in this case, the *Conference Manager* get the event from the Calendar and ask to the *Preview Service* to return the necessary data from the DB towards the client.

# Describing services: WSDL files

In order to describe the behavior and the structure of the services, we provide the WSDL files related to the *Conference Manager* and the *Preview Service* that we will implement in the next phase. The other two providers, *Maps Service* and *Calendar Service* are already existing and make use of REST protocol, therefore it is not useful provide their WSDL files.

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<definitions xmlns=*"http://schemas.xmlsoap.org/wsdl/"*

xmlns:soap=*"http://schemas.xmlsoap.org/wsdl/soap/"*

xmlns:tns=*"http://it.acl.software/services/preview"*

xmlns:wsam=*"http://www.w3.org/2007/05/addressing/metadata"*

xmlns:wsp=*"http://www.w3.org/ns/ws-policy"*

xmlns:wsp1\_2=*"http://schemas.xmlsoap.org/ws/2004/09/policy"*

xmlns:wsu=*"http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"*

xmlns:xsd=*"http://www.w3.org/2001/XMLSchema"*

targetNamespace=*"http://it.acl.software/services/preview"*

name=*"Preview"*>

<types>

<xsd:schema version=*"1.0"* targetNamespace=*"http://it.acl.software/services/preview"*>

<xsd:element name=*"previewConferenceMessageRequest"* type=*"tns:previewRequest"*/>

<xsd:element name=*"previewConferenceMessageResponse"* type=*"tns:previewResponse"*/>

<xsd:element name=*"previewMultipleConferencesMessageRequest"* type=*"tns:previewMultipleRequest"*/>

<xsd:element name=*"previewMultipleConferencesMessageResponse"* type=*"tns:previewMultipleResponse"*/>

<xsd:complexType name=*"previewRequest"*>

<xsd:sequence>

<xsd:element name=*"idConference"* type=*"xsd:string"*/>

</xsd:sequence>

</xsd:complexType>

<xsd:complexType name=*"previewMultipleRequest"*>

<xsd:sequence maxOccurs=*"unbounded"* minOccurs=*"0"*>

<xsd:element name=*"conferencesList"* type=*"xsd:string"* />

</xsd:sequence>

</xsd:complexType>

<xsd:complexType name=*"previewMultipleResponse"*>

<xsd:sequence maxOccurs=*"unbounded"* minOccurs=*"0"*>

<xsd:element name=*"conferences"* type=*"tns:ConferenceType"* />

</xsd:sequence>

</xsd:complexType>

<xsd:complexType name=*"previewResponse"*>

<xsd:sequence minOccurs=*"1"* maxOccurs=*"1"*>

<xsd:element name=*"idConference"* minOccurs=*"1"* maxOccurs=*"1"* type=*"xsd:string"*/>

<xsd:element name=*"name"* minOccurs=*"1"* maxOccurs=*"1"* type=*"xsd:string"*/>

<xsd:element name=*"abstract"* minOccurs=*"1"* maxOccurs=*"1"* type=*"xsd:string"*/>

<xsd:element name=*"urlImage"* minOccurs=*"1"* maxOccurs=*"1"* type=*"xsd:anyURI"*/>

<xsd:element name=*"urlPDFs"* minOccurs=*"1"* maxOccurs=*"1"* type=*"xsd:anyURI"*/>

<xsd:element name=*"city"* minOccurs=*"1"* maxOccurs=*"1"* type=*"xsd:string"*/>

<xsd:element name=*"date"* minOccurs=*"1"* maxOccurs=*"1"* type=*"xsd:date"*/>

<xsd:element name=*"latitude"* minOccurs=*"1"* maxOccurs=*"1"* type=*"xsd:double"*/>

<xsd:element name=*"longitude"* minOccurs=*"1"* maxOccurs=*"1"* type=*"xsd:double"*/>

</xsd:sequence>

</xsd:complexType>

<xsd:complexType name=*"ConferenceType"*>

<xsd:sequence>

<xsd:element name=*"name"* minOccurs=*"1"* maxOccurs=*"1"* type=*"xsd:string"*/>

<xsd:element name=*"abstract"* minOccurs=*"1"* maxOccurs=*"1"* type=*"xsd:string"*/>

<xsd:element name=*"urlImage"* minOccurs=*"1"* maxOccurs=*"1"* type=*"xsd:anyURI"*/>

<xsd:element name=*"urlPDFs"* minOccurs=*"1"* maxOccurs=*"1"* type=*"xsd:anyURI"*/>

<xsd:element name=*"city"* minOccurs=*"1"* maxOccurs=*"1"* type=*"xsd:string"*/>

<xsd:element name=*"date"* minOccurs=*"1"* maxOccurs=*"1"* type=*"xsd:date"*/>

<xsd:element name=*"latitude"* minOccurs=*"1"* maxOccurs=*"1"* type=*"xsd:double"*/>

<xsd:element name=*"longitude"* minOccurs=*"1"* maxOccurs=*"1"* type=*"xsd:double"*/>

</xsd:sequence>

</xsd:complexType>

</xsd:schema>

</types>

<message name=*"previewConferenceMessageRequest"*>

<part name=*"parameters"* element=*"tns:previewConferenceMessageRequest"*/>

</message>

<message name=*"previewConferenceMessageResponse"*>

<part name=*"parameters"* element=*"tns:previewConferenceMessageResponse"*/>

</message>

<message name=*"previewMultipleConferencesMessageRequest"*>

<part name=*"parameters"* element=*"tns:previewMultipleConferencesMessageRequest"*/>

</message>

<message name=*"previewMultipleConferencesMessageResponse"*>

<part name=*"parameters"* element=*"tns:previewMultipleConferencesMessageResponse"*/>

</message>

<portType name=*"PreviewPT"*>

<operation name=*"previewConferenceRequest"*>

<input message=*"tns:previewConferenceMessageRequest"*/>

<output message=*"tns:previewConferenceMessageResponse"*/>

</operation>

<operation name=*"previewMultipleConferencesRequest"*>

<input message=*"tns:previewMultipleConferencesMessageRequest"*/>

<output message=*"tns:previewMultipleConferencesMessageResponse"*/>

</operation>

</portType>

<binding name=*"PreviewBinding"* type=*"tns:PreviewPT"*>

<soap:binding transport=*"http://schemas.xmlsoap.org/soap/http"* style=*"document"*></soap:binding>

<operation name=*"previewConferenceRequest"*>

<soap:operation soapAction=*""*/>

<input>

<soap:body use=*"literal"*/>

</input>

<output>

<soap:body use=*"literal"*/>

</output>

</operation>

<operation name=*"previewMultipleConferencesRequest"*>

<soap:operation soapAction=*""*/>

<input>

<soap:body use=*"literal"*/>

</input>

<output>

<soap:body use=*"literal"*/>

</output>

</operation>

</binding>

<service name=*"PreviewService"*>

<port name=*"PreviewPort"* binding=*"tns:PreviewBinding"*>

<soap:address location=*"http://localhost:8080/preview/services/preview"*></soap:address>

</port>

</service>

</definitions>

*WSDL file for Preview service*

As we can see from the example, the WSDL file provides all information about the service, such as the types of managed data, the message involved, the end-port and the operation, the binding between abstract and concrete part. In the *Preview Service*, for example, we use several types of data; the Request and the Response message to perform the connection with DB and the other two, urlPDF to retrieve the pdf of the presentation, anyURL to access to a resource and two basic types for strings and any URL.

We provide the same description file for the *Conference Manager* that acts as prosumer in our application; the WSDL is basically the same plus several types to perform the connection to the three providers and retrieve the POIs in case of map service.

# API usage and project structure

Regarding the implementation, here below there are some details about the Google APIs used in the project and its overall structure.

**Google Maps**

To integrate this service in our project, it's enough to get the API\_KEY from Google Website to enable the retrieve of maps and in particular of POIs given certain latitude and longitude. This API return a JSON object that is used in the mobile application (the client) to display the map on the mobile phone and the related POIs. To do this, a Google account is required, and you have to request for key in free mode.

**Google Calendar**

For this service, we need the authorization to access at the personal calendar of the user through initializing the calendar in the prosumer. At the first time, the application opens a new tab in which the user must link with his google account and authorize the application to access his personal data. Then, the application download the client\_secret.json file that allow the complete access to the calendar.

**Project structure and setup**

The entire project is based on Maven and Apache CXF structure that automatically handle dependencies using the pom file in which we specify all dependencies needed for the project. Apache CXF divides the structure in two main part: the auto-generated resources and the implementation part, in which we write the business logic of our application. Going in deep on the latter, we have the src folder that contain three subfolders:  
- Java, that contain the really implementation with the all webservices involved; - Resources, that contain principally the WSDL files and all the resources for the application (such as configuration files) - Webapp, that contain the configuration files for Spring and servlet mapping Depending on the service, we change only the business logic in the business subfolder and in particular in the impl-ws path we can find the implementation of the service. To run the project, it’s enough to build all Maven projects and deploy them on Tomcat. With the update operation, Maven resolves the dependencies of the pom and build the structure (methods, interfaces) following the WSDL files that perform the mapping between the abstract and concrete resources. Once we launch the prosumer and our provider from scratch (Preview), our client can perform Restful request in order to display all necessary contents. To do this, we have set the ClientInterface servlet that allow communication with the prosumer or directly with the other providers.