Service Agents for Calendar Exchange

Magdalena Kostoska
Ss. Cyril and Methodius University
Faculty of Information Sciences and Computer Engineering
16 Rugjer Boshkovikj
Skopje, FYR Macedonia
magdalena.kostoska@finki.ukim.mk

Goran Velkoski
Ss. Cyril and Methodius University
Faculty of Information Sciences and Computer Engineering
16 Rugjer Boshkovikj
Skopje, FYR Macedonia
velkoski.goran@students.finki.ukim.mk

Krste Bozinoski
Ss. Cyril and Methodius University
Faculty of Information Sciences and Computer Engineering
16 Rugjer Boshkovikj
Skopje, FYR Macedonia
bozhinoski.krste@students.finki.ukim.mk

Sasko Ristov
Ss. Cyril and Methodius University
Faculty of Information Sciences and Computer Engineering
16 Rugjer Boshkovikj
Skopje, FYR Macedonia
sashko.ristov@finki.ukim.mk

Marjan Gusev
Ss. Cyril and Methodius University
Faculty of Information Sciences and Computer Engineering
16 Rugjer Boshkovikj
Skopje, FYR Macedonia
marjan.gushev@finki.ukim.mk

ABSTRACT

With the emerging use of electronic calendars services on the Internet the users are constantly increasing the number of calendar platforms they use and the need to have them synchronized in one place is increasing. In this paper we discuss the evolution of the electronic services for calendars, and give overview of the APIs they use, along with protocols and interfaces to these services.

We introduce an idea to create Calendar Service Agent as a software agent in order to exchange, modify and synchronize information about events from different calendar platforms. The solution is explained by protocols and patterns, as well as the structure and architecture, and platforms included.

Categories and Subject Descriptors

D.2.0 [**Software Engineering**]: General—*Standards*; D.2.11 [**Software Engineering**]: Software Architectures; D.2.12 [**Software Engineering**]: Interoperability

Keywords

Calendar protocols, calendar services, calendar interoperability

1. INTRODUCTION

Nowadays almost every person that uses electronic services also uses calendars offered by different providers. The main purpose is to schedule or share information about meetings, activities or events. Somehow every person becomes conformable using and depending upon usage of this electronic service as reminder and organizational tool.

A lot of services are offered on the market and a typical Internet user starts to use (or is forced to use) new calendars for different purposes (new work position, new group, etc.). The problem arises when at certain point one has to check several calendar services in order to synchronize all the events (even those not written in calendars) and to avoid overlapping or missing an event.

A typical architecture of different calendar usage is presented in Figure 1 showing how an user connects to multiple popular calendar services (using appropriate GUIs) in order to check the events and schedules. These activities can be also exhausting.

In this paper we give an overview of possibilities to exchange and gather different information from various calendar providers. Several conceptual models have been proposed:

- Interorganizational Intelligent Meeting-Scheduler (IIMS) [17],
- Compatible Collaborative Calendar-Server (CCS) Web Services [1] which exploit the iCalendar protocol,
- Gadget for Personal Information Integration [5] which performs data extraction and visualization, and
- Collaborator [12] which provides enterprise users with a shared workspace to support the activities of virtual teams.

BCI'12, September 16–20, 2012, Novi Sad, Serbia.

Local Proceedings also appeared in ISBN 978-86-7031-200-5, Faculty of Sciences, University of Novi Sad.

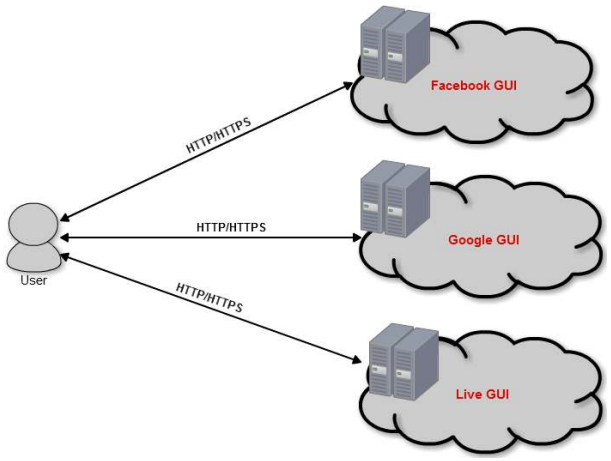

Figure 1: User connection to multiple calendar providers via their GUI.

Yet all these solution do not include most of todays' popular calendar services.

The idea about *calendar integration* is offered by some existing products, like Hipmunk and Nimble - social CRM, but they are neither stand-alone products nor they cover only some of popular calendar platforms.

Our initial motivation in this research was to exploit the calendar interoperability and to create a stand-alone software agent which joins all the popular calendar services at one place and offers the users a possibility to join all their calendar and scheduling events at one place, unlike the multiple calendar scenario shown on Figure 1. Also our goal was to exploit the existing technologies to ease the usage of multiple calendars.

The paper is organized as follows. Section 2 presents the common establish and used protocols concerning calendars and web services. Section 3 briefly describes the characteristics of the calendar platforms we have used. Section 4 gives architectural and structural overview of the new proposed solution. Finally we conclude our work and describe our future work in Section 5.

2. CALENDAR PROTOCOLS AND APIS

The need for storing calendar events and sharing standards about event storage and organization emerged in 1996 due to the big number calendars and scheduling products and their limitation to exchange information only among users within the same system. At that moment some proprietary standards existed, but there was no single and open specification. A working group of Internet Engineering Task Force (IETF) was created and this group outlined three key areas for future standardization: exchange format, interoperability protocol and access control [7].

This Calendaring and Scheduling (CalSch) workgroup over time produced few standards:

- Internet Calendaring and Scheduling Core Object Specification (iCalendar),
- iCalendar Transport-Independent Interoperability Protocol (iTIP) and
- iCalendar Message-Based Interoperability Protocol (iMIP).

Today iCalendar is widely used standard used by a large number of popular calendar products like Google Calendar [18], Apple iCal [6], Microsoft Outlook [24], IBM Lotus Notes [22] etc... Thanks to this protocol calendar files can be shared and edited by using WebDav server, which represents a RESTful interface.

In the following sections we give a short explanation about iCalendar protocol, the WebDav protocol and the REST framework. We will also explain the oAuth security protocol, which is widely used by the calendar services providers.

2.1 iCalendar

Internet Calendaring and Scheduling Core Object Specification (iCalendar) standard defines "*data format for representing and exchanging calendaring and scheduling information such as events, to-dos, journal entries, and free/busy information, independent of any particular calendar service or protocol*" [15].

It defines a MIME content type and enables exchange using several standard transport protocols (like HTTP and SMTP), different file systems and more types of transport and communication. It provides mapping of the content type to set of messages for calendaring operations [15].

2.2 WebDAV and CalDAV

The first version of the HTTP Extensions for Distributed Authoring (WEBDAV) specified "*set of methods, headers, and content-types ancillary to HTTP/1.1 for the management of resource properties, creation and management of resource collections, namespace manipulation, and resource locking (collision avoidance)*" [13].

This protocol was initially created for remote Web site authoring and it supports remote collaborative authoring of Web sites and individual documents, as well as remote access to document-management systems [31].

Calendaring Extensions to WebDAV (CalDAV) defines "*extensions to the Web Distributed Authoring and Versioning (WebDAV) protocol to specify a standard way of accessing, managing, and sharing calendaring and scheduling information based on the iCalendar format*" [14].

The CalDAV protocol provides three key features: calendar maintenance, calendar queries and calendar security [9].

2.3 REST

REST is a framework that defines Web Services with focus towards resources rather than operations. In the REST style interactions do not depend on any context stored on a server [29]. The web applications using REST web services are being build based on data and data processing. The main task is to identify resources [26]. Almost always this means cleaner and simpler service structure. By choosing them as technology, Google, Facebook and Microsoft Live, confess that REST technology is very useful for data driven applications and use the popularity of this kind of web services to encourage more developers to use their services.

The four basic design principles of REST web services are [27]:

- Use of explicit HTTP methods,
- Stateless design,
- Directory structured-like URIs and
- Transfer data using XML, JSON or even both.

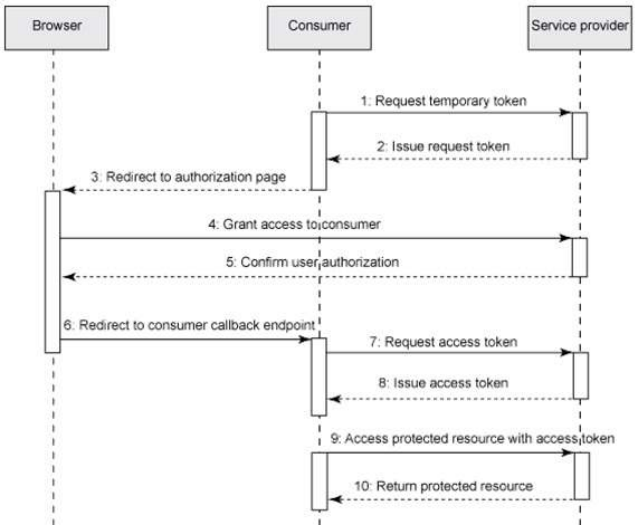

Figure 2: The so called dance of oAuth2 [2].

2.4 oAuth 2.0

The OAuth 2.0 protocol provides secure authorization in a simple and standard method from web, mobile and desktop applications. Third-party applications are using this protocol to obtain limited access over the private user data stored on different information system without knowing the user credentials.

The simplest explanation of the OAuth protocol says:

"*For most people, their car is one of their most valuable possessions, valued in tens of thousands of dollars. They are convenient places to leave our other valuables like computers and clothing. Yet we are sometimes required to give them to a parking attendant or valet whom we've not met before. A valet key solves the problem - it's an access token with limited rights that can operate the vehicle but not grant access to the trunk or glove box.*" - Eran Hammer [21].

User credentials are replaced with single text string called Access Token entrusting the client (third-party app) with limited access over the user data. For obtaining access token user grant is needed.

Validity of the token expires after certain amount of time however, if needed third-party application can be de-authorized and the generated token will be non-valid. Different platforms encourage different access token life times based on their security projections.

There are three types of tokens: BEARER TOKEN, MAC TOKEN and SAML. Services that we were using have implemented the OAuth protocol using the BEARER TOKEN which actually is big randomly generated number. Implementation of OAuth protocol which consumes BEARER TOKEN has to encrypt all service calls using SSL encryption.

Figure 2 depicts the so called oAuth dance is presented with a simple sequence diagram representing the oAuth 2.0 protocol way of work.

3. OVERVIEW OF THE CALENDAR SERVICE AGENTS

In this section we overview of the calendar platforms used in this research and their evolution.

3.1 Overview of the Included Platforms

The calendar service agent is working with a variety of platforms constructed by the social networking prodigies such as Facebook and Google followed by Microsoft. Their calendar services are popular and it's only natural that clients would want them easy accessible altogether.

In last 10 years Google, Facebook and Microsoft have created their own APIs and services regarding their calendars in order to create the best possible solution for their clients and the developers working with their platforms. By employing different technologies and techniques they all converged to probably, the best solution nowadays. This is the reason behind the fact that all the platforms i.e. Facebooks Graph API, Googles Callendar APIv3 and Live Connect API include communication by using REST and authenticate external application with oAuth 2.0 protocol. Besides the functional similarities, the platforms clearly differ as they all employ different information systems.

An essential difference between the companies and their APIs is in their business logic. Google divides their APIs in spite of Microsoft Live and Facebook who use one API to share everything. The division of APIs means that the client produces API dedicated access token whereas Facebook and Microsoft Life in order to limit their clients only to certain privileges generate different access tokens using the same API.

Another obvious difference between APIs is the format of the request and response messages, although they are all using REST web service and communicate using basic HTTP verbs.

3.2 Platforms Evolution

Platforms are subjects to constant improvement and evolution. During 2011 majority of the platforms started to deprecate the OAuth 1.0 protocol with his known security flaw. Main purposes of the evolution from 2011 were concentrated on enhancing the security of the APIs. However, the evolution is still ongoing and the new trends are demanding standardization of the APIs which will result in decreasing the learning curve for the APIs. The Calendar Service Agents had to evolve together with the changes of each platform.

Since October 2011 Facebook platform migrates to OAuth 2.0 and completely removed former Legacy Connect Auth [11]. Following Facebook, Google and Microsoft also started using OAuth 2.0 as basic authorization protocol [19], [25].

Enhancing the security Facebook announced that from October 2012 long-lived tokens which never expired will be replaced with tokens which are valid up to 60 days [10].

Google in the version 3 of the Calendar API will replace GData response format with JSON data objects In order to satisfy clients needs Google decided to keep GData response format active, minimizing the clients costs for migration [20]. GData response format required additional client API to be installed in order to successfully retrieve data and installation of this client wasn't trivial task.

4. SOFTWARE AGENT ARCHITECTURE

In this section we describe the architecture of the new proposed software agent for calendars. Due to popularity of web services, the calendar service agent is in fact a web service.

For our research we have developed a prototype using Java EE. Data for the web service is being stored in the MySQL database. The calendar agent is constructed as a RPC style web service using SOAP messages for communication. Data in the messages are structured as JSON strings because of JSONs' clean structure and in order to make the web service easy to use and integrate. The whole calendar agent architecture is depicted in Figure 3.

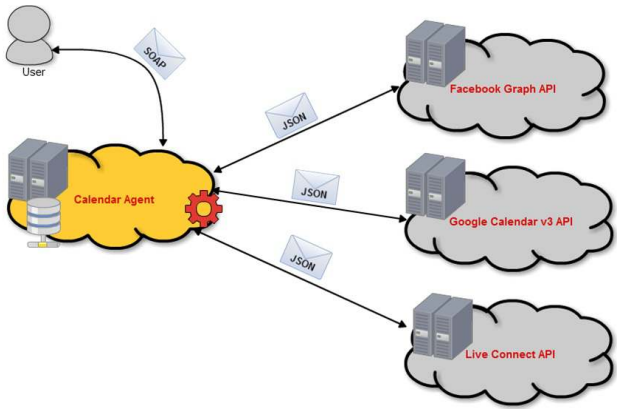

Figure 3: Calendar Agent architecture.

The calendar agent was implemented using the "facade" design pattern by integrating and wrapping up three other private subagents that can't be accessed from an external agent. Each subagent has assignment to execute a task on a particular platform and to store the results from the task in the database. Clients can only access the calendar agent by sending authentication details wrapped in a single JSON message. The message has to be SSL encrypted and its contents vary depending on the task being executed. If the task is successful then the results are stored in the database and sent back to the user like JSON string.

Figure 4 depicts the structure of the calendar agent. Private factory methods, that generate instances of subagents, construct the calendar agent. Each subagent respectively implements entire logic for executing tasks for the dedicated platform. Scalability for the calendar agent is being ensured by following the open/closed object orientated principle for subagent development.

The number of accounts the clients can have is unlimited. Registration of each account requires valid access token that will be stored in the database of appropriate calendar agent.

Functions implemented in the current calendar agent are:

- Fetch events,
- Create event,
- Update event and
- Delete event.

5. CONCLUSION AND FUTURE WORK

This paper shows how to exploit and compose together the possibilities of calendar interoperability offered by the existing and popular calendar solutions. We have succeeded in our intention and we have created Calendar Agent Services as a one check point for the most popular calendars the user have today. With this application the user don't have to

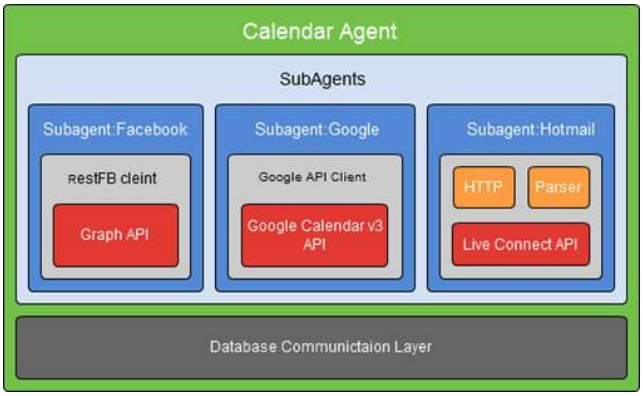

Figure 4: Calendar Agent structure.

check all the different calendar services he/she uses on their native platforms, but to have them all in one place. This solution is service-oriented, easily scalable and modifiable and it includes the popular calendar services. The whole solution is built with web services, so the interface can be changed very fast and it is easily adoptable and maintainable to different device platforms.

Calendar Service Agents is not the only calendar synchronization and exchange calendar solution. There are a lot of Desktop calendar solutions like Desktop Calendar [28] and Desktop iCalendar [8] but they are restricted in platforms and can be used only on the pre-installed computer. VueSoft have developed VueMinder Calendar [30] but this solution supports windows platform only, does not support Facebook synchronization and beside the regular price the user has to pay additionally to have synchronization in both directions. Also the Blackberry Default Calendar (CICAL) [3], besides the limitation to the platform have issues with Facebook synchronization and by a lot of user have been declared as a non-user-friendly product. There a also a few Android products like Smooth Calendar [4], Fliq Calendar [23] or CalDAV-Sync [16] but they are also limited only to the Android platform and not all them support all the popular services.

Our solution it is unique, it is based on web services, offers more adaptability since it can be reached from any browser (including smart phone browsers) and includes today most popular calendar services.

Calendar Service Agents is a project that constantly evolves and adapts. In the last year we had to make a lot of changes due to change of platforms by different calendar providers, changed APIs or authentication protocols. At the moment we provide user interface that can be used via browser.

Our next milestone is to extend this project with mobile applications for Android and iOS and to include other popular calendar services. We also want to include oAuth 2.0 authentication for our agent and to create REST web services for our project, beside the standard RPC solution. Our further future work include extending the research to wide-range cloud-based SaaS calendars interoperability.

6. REFERENCES

[1] W. W. Ahmet Fatih Mustacoglu and G. Fox. Internet calendaring and scheduling core object specification (icalendar) compatible collaborative calendar-server

(ccs) web services. In *International Symposium on Collaborative Technologies and Systems*, pages 12–17, 2006.

[2] Z. Bi and X. Yu. Add oauth authentication support to httpclient, Dec. 2010.

[3] Blackberry. How to merge multiple calendars into one calendar on the blackberry smartphone.

[4] C. Cedergren. Smooth calendar.

[5] C.-M. L. Chia-Hui Chang, Shih-Feng Yang and M. Kayed. Gadget creation for personal information integration on web portals. In *IEEE International Conference on Information Reuse and Integration*, pages 469–472, 2008.

[6] D. Crow. AppleâĂŹs ical and ical/vcal format.

[7] F. Dawson. Emerging calendaring and scheduling standards. *Computer*, 30(12):126–128, Dec. 1997.

[8] Desksware. Desktop icalendar – internet desktop calendar.

[9] L. Dusseault and J. Whitehead. Open calendar sharing and scheduling with caldav. *Internet Computing, IEEE*, 9(2):81–89, Mar.–Apr. 2005.

[10] Facebook. Authentication, 2012.

[11] Facebook. Completed changes, Jul. 2012.

[12] M. M. Federico Bergenti and M. Garijo. Collaborator – enabling enterprise collaboration through agents. In *13th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, pages 41–46, 2004.

[13] I. E. T. Force. Http extensions for distributed authoring – webdav, Feb. 1999.

[14] I. E. T. Force. Calendaring extensions to webdav (caldav), Mar. 2007.

[15] I. E. T. Force. Internet calendaring and scheduling core object specification (icalendar), Sept. 2009.

[16] M. Gajda. Caldav-sync beta.

[17] C. Glezer. A conceptual model of an interorganizational intelligent meeting-scheduler (iims). *Journal of Strategic Information Systems*, 12:47–79, 2003.

[18] Google. Import events from icalendar or csv files.

[19] Google. Oauth 2.0 playground: Open to developers!, Nov. 2011.

[20] Google. Migrating to google api java client, Jun. 2012.

[21] E. Hammer. Explaining oauth, Sept. 2007.

[22] IBM. Ibm lotus notes 8.5 icalendar: Interoperability, implementation, and application.

[23] Mark/Space. Fliq calendar.

[24] Microsoft. View and subscribe to internet calendars.

[25] Microsoft. Windows live developer platform updated with oauth 2.0 support and more, Jun. 2011.

[26] A. R. Mikko Hartikainen, Markku Laitkorpi and T. Systa. How to drill down to rest apis: Resource harvesting with a pattern tool. In *13th IEEE International Symposium on Web Systems Evolution (WSE)*, pages 135–140, 2011.

[27] A. Rodriguez. Restful web services: The basics, Nov. 2008.

[28] T. Software. About desktop calendar.

[29] H. S. Takeru Inoue, Hiroshi Asakura and N. Takahashi. Key roles of session state: Not against rest architectural style. In *IEEE 34th Annual Computer Software and Applications Conference (COMPSAC)*, pages 171–178, 2010.

[30] VueSoft. Vueminder feature overview.

[31] J. Whitehead. Webdav: versatile collaboration multiprotocol. *Internet Computing, IEEE*, 9(1):75–81, Jan.-Feb. 2005.