Research and Implementation of CalDAV-based Enterprise Calendar Service

Erdong Ma[1,a], Yu Zhang[2,b]

Information Technology Center, Tsinghua University, Haidian District, Beijing, China

[a]med@tsinghua.edu.cn, [b]zhangyuthu@tsinghua.edu.cn

Keywords: Enterprise Calendar; Calendar; WebDAV; CalDAV; Event; Todo; Cross-domain; Same-origin policy

Abstract. Calendar service protocol stack consists of four protocols: WebDAV, CalDAV, iCalendar and jCal. The basic functions of calendar service include personal planning, schedule sharing, events reminder, invitation and data synchronization among platforms or devices. Calendar service has a lot of commercial and open-source implementations such as Google calendar, Apple calendar, DAViCal, SabreDAV and Radicale. Role-based authorization and data synchronization are the two key technologies discussed. Problems like data cleansing, batch operation interfaces and cross-domain issues are given concerns, too.

The Introduction of Calendar Service

Calendar Service Protocols. Calendar service is a protocol group made up of four protocols: WebDAV, CalDAV, iCalendar and jCal. WebDAV locates at the bottom of the whole protocol stack. It allows users to manage files on the server [1]. CalDAV is an extension to WebDAV, which defines a standard way of accessing, managing, and sharing calendar information based on the iCalendar format [2]. jCal defines a JSON format for iCalendar data [3]. As a commonly used data interchange format, JSON has several advantages: lightweight, text-based and language-independent etc.

Content of Calendar Protocols. Calendar service has four core concepts: calendar, event, todo and journal. Calendar is a container, which includes events, todos and journals. A person can have multiple calendars, such as personal calendars, work calendars, study calendars etc., to organize different types of events and todos; Event may be one activity lasting for a period of time or something you want to do. Generally, an event should have attributes like name, location, start time and end time. We can also specify an event as busy or idle, so that you know whether a certain event can be parallel with other events or not. This is particularly important when the schedule is shared among users. Todo generally has a plan time and the actual completion time, it emphasizes more on the concept of time point than time period. Journal does not take time, it plays a role of organizing and taking memos. Many commercial and open source calendar applications have not implemented journal function because it overlaps with events and todos in concept [4].

The basic functions of the calendar service include the following four aspects [5]:

• Personal schedule maintenance

Users can create multiple calendars, add, modify, or delete events, todos in each calendar.

• Share schedule

Users can share their calendar to colleagues, family and friends, and view calendars shared by other people. This will facilitate coordination among a group of people or a team [6].

• Send the invitation and accept reply

After creating a user event, you can invite others to participate. Invitees can reply to event invitations via e-mail or calendar.

• Synchronization cross-platform and cross-device

Since following the same calendar protocol, the calendar can therefore easily synchronized between web, desktop applications, mobile phones and other devices, so that we can get information anytime, anywhere.

Open-source Implementations of Calendar Service

Calendar service has a lot of implementations, including both open-source software and commercial applications. The most popular commercial implementations are Google and Apple calendars. Google calendar offers two kinds of clients: web version and mobile version [7]. Apple calendar include desktop and mobile clients too, but only run on Mac OS X and iOS operating systems [8]. After years of development and iteration, these commercial implementations could provide us with an useful reference in all aspects like function, design, interface and human-computer interaction.

Open source clients can generally be divided into 3 types: desktop, mobile, and Web. There are just a small number of mature desktop clients, for example GNOME's Evolution and Mozilla Foundation's Lightning. In recent years, with the rapid development of mobile devices, a large number of excellent calendar clients emerge in Android and iOS platforms. Desktop and mobile clients are basically mature products, most of which are closed source. They all have good standard protocol support, but can hardly be customized. Web client bases on HTML / CSS / Javascript technology, runs in the browser in the form of source code. Therefore, the web version of the client is the only client types that could support secondary development.

Server-side calendar service has many mature implementations. Most of them are developed in PHP, Python or other dynamic languages, using MySQL, PostgreSQL or other open source databases. The popular projects are DAViCal, SabreDAV and Radicale. If we want a standard server-side implementation following CalDAV protocol, most of them are equal to the task. If we want to make secondary development, a careful choice should be made depending on the requirements of your project.

Main Features of Enterprise Calendar Service

Standard calendar service is designed for individuals, while organizations and companies are the main targets of enterprise calendar service. These two kinds of calendar services differ in many aspects according to their usage scenarios.

Standard calendar service has an internet-oriented, loosely organized structure. End users have full control of their own calendars and could authorize other users to read/write their personal calendar and format an authorization network. Enterprise calendar is organization-oriented. Unlike standard calendar, it has tree structure naturally. Enterprise calendar could still be shared between users, if we take this into account, authorization structure of enterprise calendar is hierarchy tree structure vertically and net structure horizontally.

Calendar components mainly consists of events and todos. In standard calendar service, all of the information is generated by individuals. End users are information producers and consumers at the same time. The data of Enterprise calendar mainly comes from the enterprise's own business systems. For instance, they tend to draw up the whole business plans at the beginning of every year, due to the huge staffs and resources involved, this kind of plans, almost never change. After the enterprise publishes its strategy, every department will has their own arrangements. Individuals within the enterprise could read part or whole of information according to their authorities, the so called personal events could just be arranged among the lapses of the settled enterprise events.

Implementation of Enterprise Calendar Service

Role-based Authorization. Standard calendar has user-based authorization system while business systems usually have role-based authorization. Mapping data of roles and users, could be maintained either in the business side or the calendar side.

If you were to maintain the role data in business systems, authorization granularity should be your main concern. The granting and revoking of R/W access privileges are based on specific calendars, so calendar is the smallest granularity. The rules describing which role could read or write which calendar, are pushed to calendar server by business systems. Calendar server does not care about the business logic, it just needs to know the relationships among people and calendar. If you were to maintain the role data in the calendar server, the calendar needs to know the current user role which can be mapped to local calendars. This greatly reduces the difficulty to dock the calendar service to business systems, but also increases the maintenance workload of local calendars. On the other side, as not specified in the calendar service protocol, this solution requires secondary development on the basis of the open source implementation, and has high technical barriers.

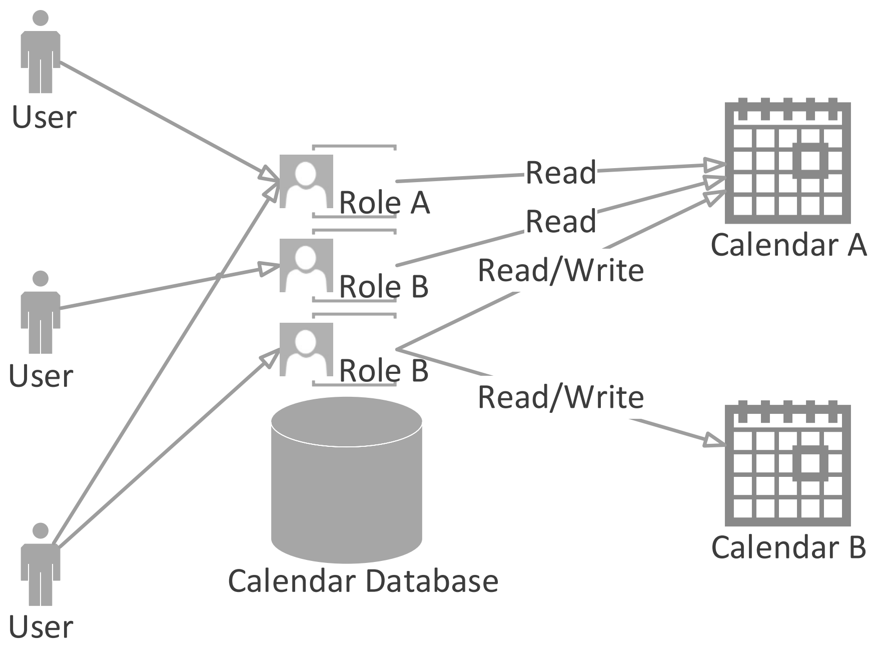

Fig. 1 Role-based authorization mapping

Data Synchronization. Timing synchronization is data synchronization between enterprise information systems and calendar services at regular intervals. The advantage of this approach is simplicity - we can avoid the complex business logic of information systems, data is read directly from the databases of existing information systems, then imported to calendar service regularly. The disadvantage is that you could never achieve real-time, although the incremental timing synchronization could greatly shorten the time intervals and improve synchronization frequency to reach a quasi-real time effect.

Timing synchronizations are automatically triggered according to specific business requirements and deployment environment. Data of calendar service generally comes from multiple business systems, and the data synchronization plan is a fusion of multiple timing synchronization patterns (whole set or incremental). Among business systems, synchronization sequence, manner and frequency are all problems to be solved. So we need to establish a unified data adapter to manage the synchronizations. No data adapter is required in real-time synchronization scenarios due to its high timeliness requirements, in which case business systems often interact directly with the calendar service.

Real-time synchronization is the ideal solution. As mentioned earlier, the “Data Sharing Mode” is a complex job. In practice, timing synchronization often coexists with real-time synchronization.

Large data sets, frequently changing, real-time synchronization, it is extremely difficult to meet all these requirements at the same time, and it is not the focus of this paper. The more common scenario is to meet two of the three requirements, and to make trade-offs in the last one. Before designing the architecture, detailed analysis should be applied to the data and usage scenarios.

Real-time synchronization has two patterns logically: “Push” and “Pull”. “Push” pattern means that business systems push the latest data to calendar service in real time when the business data changes. This requires adding business triggers at key nodes of business systems. Business trigger introduces a deep integration scheme on behalf of the business logic. When data in business system changes, a request will be triggered immediately to push this change to calendar service. Considering the complexity of business logic, you may have a large number of trigger points. Therefore, this solution is suitable to new established business systems. Calendar module should be taken into account as an embedded module from the beginning, and should also be an important concern in the future expansion of the business systems.

“Pull” pattern means that the calendar service askes business systems for the latest data right after the request of current user. This kind of requests are often triggered by actions like “login” or “click”. The advantage is obvious. First, the amount of data is greatly reduced. The whole set of enterprise data is very large, but the data size of a single user may be many orders of magnitude smaller. Secondly, the business logic generally bases on the user point of view, we can reuse a lot of them and treat the existing business logic as a black box.

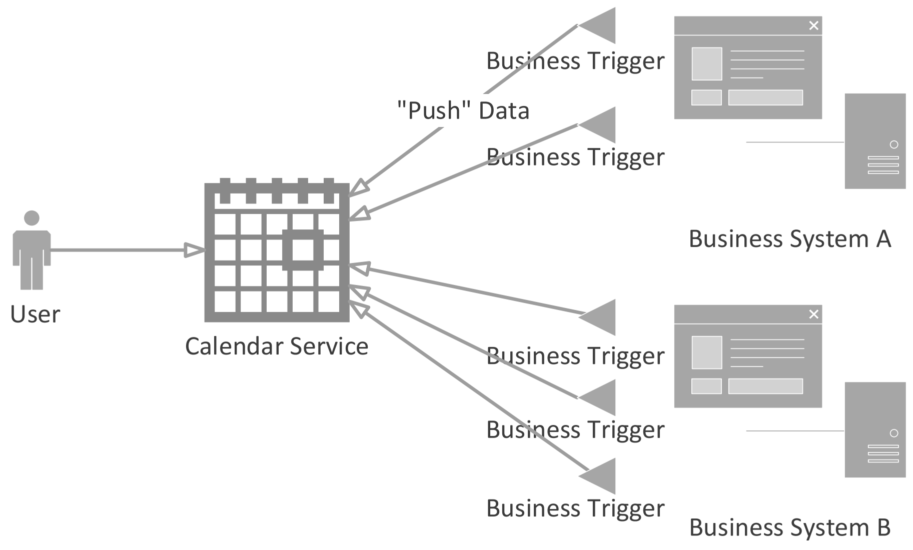

Fig. 2 “Push” pattern of data synchronization

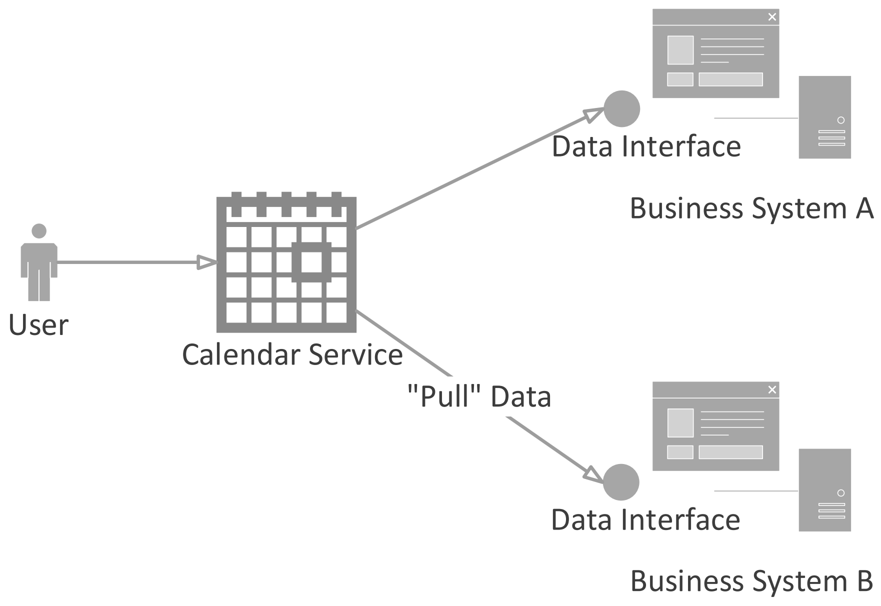

Fig. 3 “Pull” pattern of data synchronization

Cleaning Up and Withdrawal Mechanisms. There are two ways to handle expired data generally: “remove” or “archive”. Daily operations of information systems will generate a lot of data, if left untreated, continued expansion of the data would impacts the service quality of information systems, and would also results in a tremendous waste of IT resources. A qualified information system, should not allows the data to grow unlimitedly, but rather working with a dynamic data flow model.

As mentioned earlier, the calendar service can be divided into two categories. In first category, the data comes from existing business systems. For this category, the calendar service is a standardized way to show data, but not the data source. Therefore, the reasonable approach is to keep pace with the real data source. After all, the enterprise information systems run as a whole, calendar service should be well adapted to integrate into this whole system. If the data was removed in business systems, it should not exists in calendar service either. For data generated by the calendar service itself, if is confirmed invalid, we archive them generally.

Key Technologies of Enterprise Calendar Service

Design of Batch Operation Interfaces. Both business systems and standard calendar service need to be modified if you want to synchronize data among them, and these modifications differ depending on the timeliness requirements. Whatever you want, bulk operations should be added to the standard calendar service interfaces. As noted above, the current calendar service agreements were established without considering the scenarios of enterprise applications. In order to meet the new requirements, agreements should be extended.

Bulk operation interfaces pack multiple “add”, “delete” and “edit” operations in one request, one operation for one resource. Calendar server performs these operations one by one, and returns a batch of results for every operation one-time. The http request method is POST, namespace for all new labels is MD (MultipleDAV). Outermost label is MD:multipost, while each operation is placed inside MD:resource. MD:put, MD:post and MD:delete represent “add”, “edit” and “delete” operations respectively, while D:href tag identifies the operating targets. The result consists of the D:multistatus and D:response tags. Inside D:response tag, D:href identifies the operating object, D:status contains state code [9]. The sample codes are demonstrated below.

```
<MD:multipost xmlns:D="DAV:" xmlns:MD="Error, invaid url reference!"
xmlns:C="urn:ietf:params:xml:ns:caldav">
    <MD:resource>
           <D:href>/calendars/xxx/default/1.ics</D:href>
           <MD:put/>
      <D:set>
        <D:prop>
          <C:calendar-data>...icalendar event...</C:calendar-data>
        </D:prop>
      </D:set>
     </MD:resource>
     <MD:resource>
           <D:href>/calendars/xxx/default/2.ics</D:href>
           <MD:post/>
      <D:set>
        <D:prop>
          <C:calendar-data>...icalendar event...</C:calendar-data>
        </D:prop>
      </D:set>
     </MD:resource>
     <MD:resource>
      <D:href>/calendars/xxx/default/3.ics</D:href>
      <MD:delete/>
     </MD:resource>
</MD:multipost>
```

Fig. 4 Request XML of bulk operations

```
  <D:multistatus  xmlns:D="DAV:"  xmlns:MD="http://calendar_example.com/namespace/"
xmlns:C="urn:ietf:params:xml:ns:caldav">
  <D:response>
    <D:href>/calendars/xxx/default/1.ics</D:href>
    <D:propstat>
      <D:prop>
        <D:getetag>"62bc34ad2dfbc4cd60133aa3133f7c3f"</D:getetag>
      </D:prop>
      <D:status>HTTP/1.1 200 OK</D:status>
    </D:propstat>
  </D:response>
  <D:response>
    <D:href>/calendars/xxx/default/2.ics</D:href>
    <D:propstat>
      <D:prop>
        <D:getetag>"695f5bbc005682cbcf5c81c66d9d783e"</D:getetag>
      </D:prop>
      <D:status>HTTP/1.1 200 OK</D:status>
    </D:propstat>
  </D:response>
  <D:response>
    <D:href>/calendars/xxx/default/3.ics</D:href>
    <D:status>HTTP/1.1 403 FORBIDDEN</D:status>
  </D:response>
</D:multistatus>
```

Fig. 5 Response XML of bulk operations

Cross-domain Problem. As we all know, because of the limit of source policy (Same-origin policy), JavaScript can’t get or operate documents from other domains. Here the “domain” consists of the communication protocol, domain name and port [10]. In the practical deployment condition, cross-domain problem could not be avoided due to communications across multiple business systems and calendar service.

Two solutions exist in web development about cross-domain problem: client-based and server-based methods. Client-based solution is based on PostMessageTransport or script / iframe tag. The former one uses Window.postMessage function in javascript and could send messages to another domain. This function works in Internet Explorer 8+ and all other modern browsers. Scirpt and iframe tag in HTML can refer resources of another domain, this feature could also be used to complete cross-domain operations. The popular jsonp technic is implemented on the basis of the cross-domain feature of script tag. The combination of these two technologies fit most situations in web development, but they are not suitable to calendar service scenario. Cross-domain solutions above could not send request with REPORT or PROPFIND http methods, which appear in WebDAV and CalDAV standards as extensions to HTTP specification. It is important to note that Microsoft Internet Explorer 8 can’t send REPORT requests through its XMLHttpRequest object, so even in the absence of cross-domain issue, the Microsoft Internet Explorer 8 and below versions are unable to work with calendar applications.

Reverse proxy is the server-side solution. If calendar service client was embedded in business systems, we would configure reverse proxy in web server of business system to point to the calendar service. In this case, all browser requests are sent to business systems, and requests matching specific rules will be forwarded to calendar service. In other words, the clients and calendar service communicate indirectly through reverse proxy mechanism of web server, and this process is completely transparent for the browsers, so that you can conduct all communications between clients and server in the same domain of business system.

Conclusions

In recent years, with the rapid development of mobile devices, electronic calendar are spread quickly. However, the development of enterprise calendar is just unfolding. Considering the different usage scenarios, enterprise calendar and standard calendar have many differences. In order to meet the specific requirements of enterprises calendar, we need secondary development on the basis of existing open source projects. Role-based authorization and real-time data synchronization both have many intricacies and peculiarities. Taking into account the difficulties of real-time data synchronization, for scenarios that do not require timely data, timing synchronization is enough. The difficulty of timing synchronization is to extend the CalDAV protocol by adding interfaces of bulk operations following the rules of individual component operations. As the number of business systems increasing, synchronization scheduling will turn out to be a problem finally. The combination of real-time and timing synchronization is the recommended practice in reality.

References

[1] Dusseault, L. Ed. HTTP Extensions for Web Distributed Authoring and Versioning (WebDAV) RFC4918, 2007. http://tools.ietf.org/html/rfc4918

[2] Desruisseaux, B. Internet Calendaring and Scheduling Core Object Specification (iCalendar), RFC5545, 2009. https://tools.ietf.org/html/rfc5545

[3] Kewisch, P., Daboo, C., Douglass, M. jCal: The JSON Format for iCalendar. RFC7265, 2014. https://tools.ietf.org/html/rfc7265

[4] Daboo, C., Desruisseaux, B. and Dusseault, L.: Calendaring Extensions to WebDAV (CalDAV), RFC4791, 2007. https://tools.ietf.org/html/rfc4791

[5] Google Inc. Welcome to the Google Calendar Help Center. https://support.google.com/calendar

[6] Yoshinari Nomura, Yuya Murata, Hideo Taniguchi, Masakazu Urata, Shinyo Muto. Bring Your Own Calendar: A CalDAV-based Virtual Calendar System. 2013 Eighth International Conference on Broadband, Wireless Computing, Communication and Applications, pp. 551-556.

[7] Google Inc. Google Calendar. https://www.google.com/calendar/

[8] Apple Inc. Apple Calendar. https://www.apple.com/cn/osx/apps/#calendar

[9] Lisa Dusseault, Jim Whitehead. Open Calendar Sharing and Scheduling with CalDAV. IEEE Computer Society. Mar.2005. pp. 81-89.

[10] World Wide Web Consortium. W3C Web Security. https://www.w3.org/Security/wiki/Same_Origin_Policy