
# Bachelor Theses

**Synchronization between Funambol Server and Google Calendar**

Made by:
Fábio Ruivo Ferreira
xferre02@fit.vutbr.cz

Tutor:
Ing. Jaroslav Ráb
rabj@fit.vutbr.cz

# Index

# 1. Abstract

This project consists in a plug-in that will make the synchronization between the Funambol Server and the Google calendar.

# 2. Key Words

**Funambol** – Company that provides the Funambol server.[2]

**Funambol Server** – Open source server produced by Funambol, allows the synchronization of cell phones, PDAs, outlook, etc.[4]

**SIF-E** – Type of event used by Funambol Server to save the calendar events.[3]

**Google Calendar** – Calendar provided by Google, this calendar is a web application that allows scheduling events in a calendar.[6]

**Google Calendar API** – Google provide an Application Programming Interface (API) that allows the manipulation of the calendar.[6]

**RFC 2445** – Internet calendaring and scheduling core object specification. [9]

**vCalendar** – Standard type of a calendar event, used by Google calendar.[10]

**iCalendar** – Upgrade of the vCalendar, is like a vCalendar 2.0.[10]

# 3. Introduction

The development of this project was split in two different parts, first the understanding of the technologies in this project, mainly Funambol and Google Calendar API and then the code development for the synchronization.

I tried to split the logic of the code, the division was made between the framework and the plug-in modules, the framework have all the data access and the plug-in have everything to make the synchronization.

This project was developed in the Operation System (OS) *Ubuntu* (*unix*/*linux*)[13], this OS was installed in a virtual machine, *Vmware [12]* and all the development was made using the Net Beans IDE.

# 4. Technologies

## 4.1. Funambol Architecture

The synchronization architecture is based upon a *SyncSource* that wraps a set of items to be synchronized. Any type of data (files, database tables, calendar events and so on) can be synchronized, but there must be a proper *SyncSource* for each type, able to extract and store the data for a real data store. [3]

In the figure 1 and 2 is represented the order that the methods will be called for the synchronization. There are two types of synchronization:

- Slow sync: synchronize all the data, normally is used only when is the first sync.
- Fast sync: check if there were changes and synchronize only those changes.

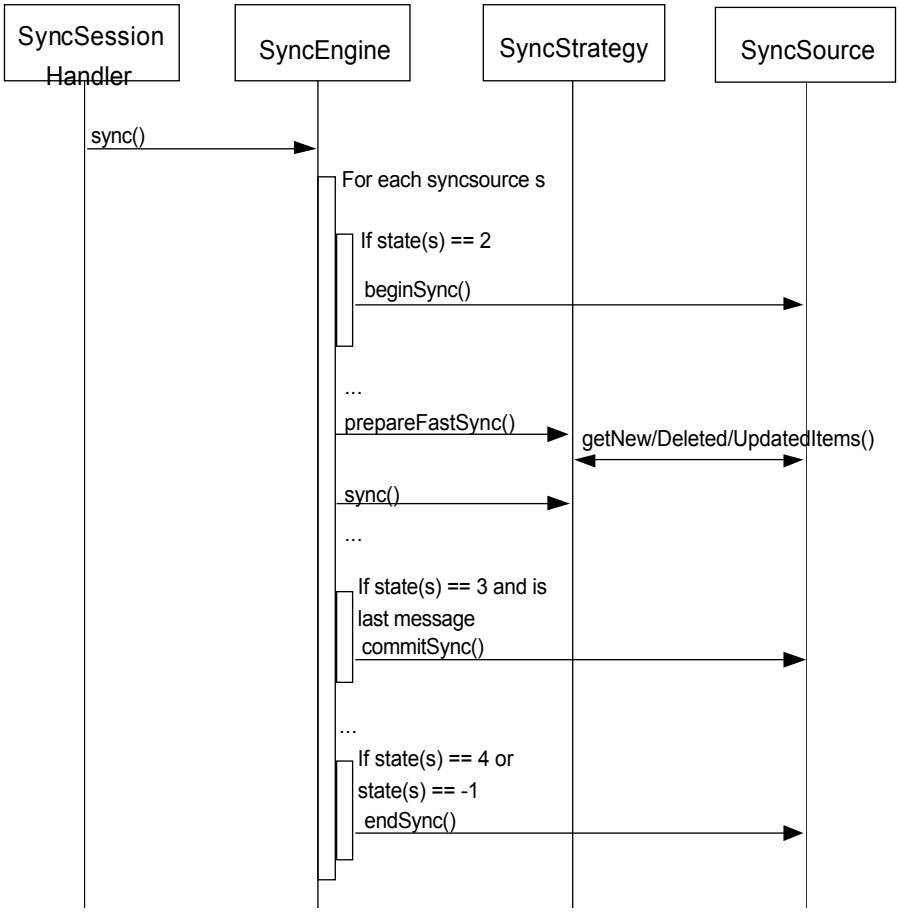


*Figure 1 - Sequence diagram of a successful fast synchronization [4]*

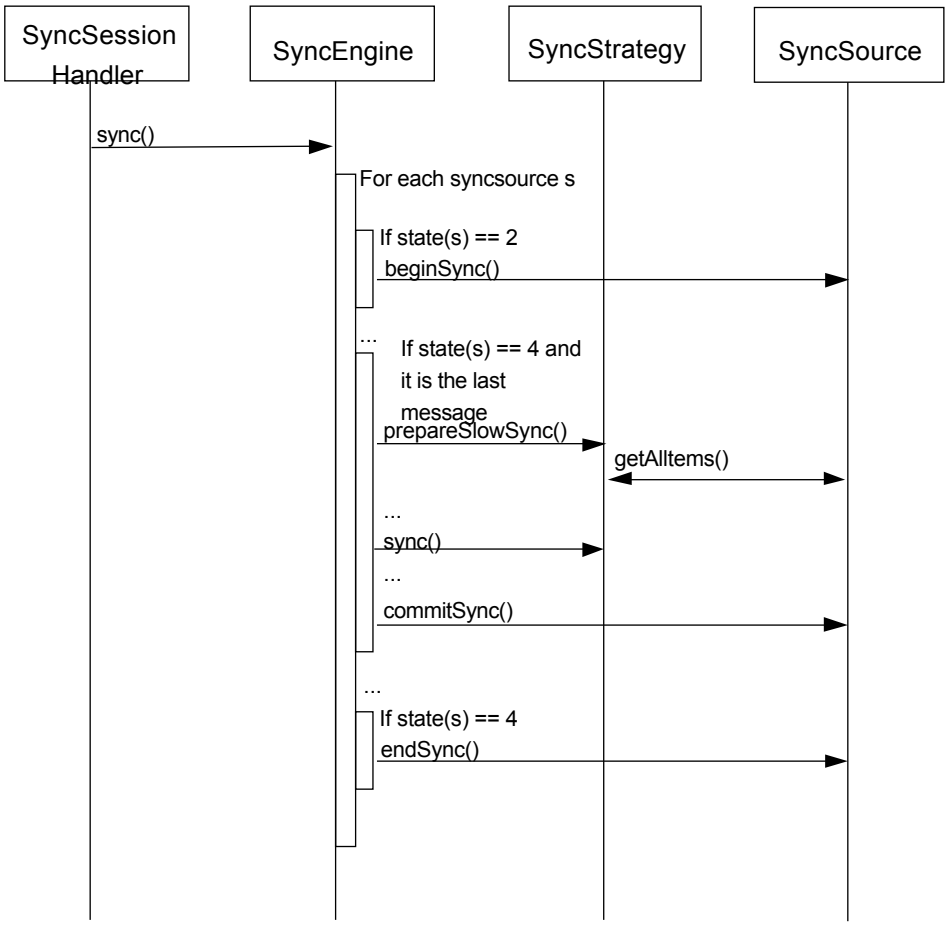


*Figure 2 - Sequence diagram of a successful slow synchronization [4]*

There was a project in Funambol developers that synchronize Google contacts and Funambol server, the logic of the sync of this project is the same, so it was an easier way to understand the solution. This project can be seen in the VMware image, in the folder */home/fabio/project/funambol-gmail only contacts/*, this is the version in svn repository and the build is made in the same way as this project, explained in chapter 6.

I grab this project and start to implement my solution. So the final solution will permit sync contacts and calendar in the same application.

The user interface is the same and is used some other parts too, for example *GoogleContactManager* that have the access to the database of the Funambol.

**SIF-E**

The SIF-E is the Funambol type of calendar event, which represents an event scheduled for a day at a particular time or a series of days at a particular time.

Here we can save all the normal information of a calendar event. The Funambol API provides some classes that permit some conversion and manipulation. [3]

## 4.2. Google Calendar API

Google Calendar allows client applications to view and update calendar events in the form of Google Data API feeds. Your client application can use the Google Calendar Data API to create new events, edit or delete existing events, and query for events that match particular criteria. [6]

### 4.2.1. icalendar (ical) or vcalendar

To represent the calendar events the Google API uses the *RFC 2445* standard types *ical* or *vcalendar*. The *ical* type is just a *vcalendar* 2.0, so to do this project I focus in the *vcalendar* only. For example one improvement that an *ical* had is the propriety on the event that you can check has available or busy. [10]

## 4.3. Apache Maven

Have I said earlier this project is the continuation of the contact sync, and that project is using the Maven tool, I get familiar with it and I continue to use.

Maven is a standard way to build the projects, a clear definition of what the project consisted of, an easy way to publish project information and a way to share JARs across several projects. [11]

I used net beans as IDE and there is a Maven plug-in for net beans getting the development really easier.

# 5. Development

## 5.1. Google Calendar Framework

### 5.1.1. GoogleCalendarModel

This class extends the class *Calendar* of the Funambol and it will permit that instead of a simple *Calendar* we can associate an ID to identify the calendar, a *TimeStamp* to save the time of the last update and *CalendarEventEntry* that is a object from the Google API where is saved the information of the Google event.

This class is to support the logic of the information of each calendar/event that come/goes to the Google calendar.

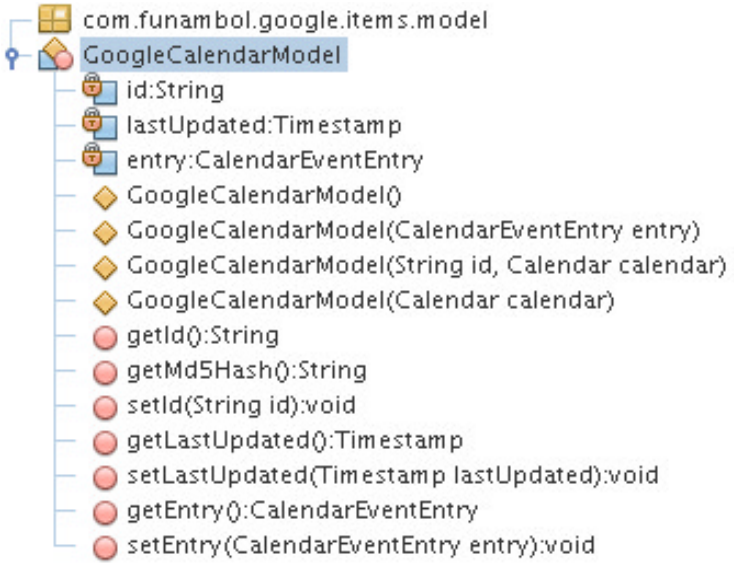


*Figure 3 - Information about GoogleCalendarModel*

### 5.1.2. CalendarRecurrence

There are two types of events, simple and recurrent, a recurrent event is the one who repeat every day or every week for example, and with this type of event we have to be more careful. (9)

Example of a recurrent event String:

```
DTSTART;VALUE=DATE:20080501
DTEND;VALUE=DATE:20080502
RRULE:FREQ=WEEKLY;WKST=SU;BYDAY=TH;UNTIL=20080529
```

It means that every Thursday between 01/05 and 29/05.

The Recurrence in the Google calendar side is assured by the type Recurrence provided in the Google API, but this class was needed because when we want to convert a Google recurrent event into a *GoogleCalendarModel* Event we need to make this conversion by "*hand*".

The constructor of this class receives a String that have all the recurrence information, the only thing that is necessary to do is to parse this information and create a event based on this information.

To parse this information is used an implementation of the *RFC 2445* that is used to parse recurrence rules. While the string is parse is put into the *RFC* objects some information, and is created in the same time the event type of Event that is an object provided by the Funambol Architecture.

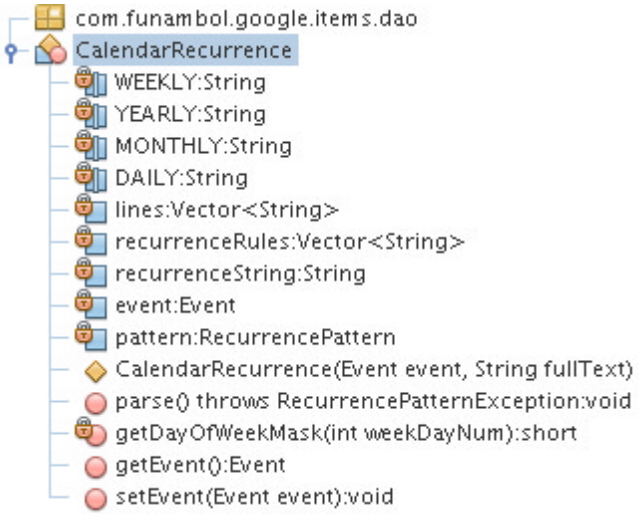


*Figure 4 - Information about CalendarRecurrence*

### 5.1.3. GoogleCalendarDAO

This is the data access object (DAO) that will make all the operations in the side of the Google calendar, it contain methods to access and control the information inside the Google calendar.

In this class we have some important fields:

**listOfEvents**

This field is a *Hashtable <String,GoogleCalendarModel>*, it will save all the events in the calendar, the *Hashtable* have two fields, a String id that is associated to the event and other that is the related event type of *GoogleCalendarModel*.

**service**

This field is a *GoogleService* that is provided by the Google API and it gives support for authentication, without this we cannot do any operations in Google calendar.

**timeZone**

This field is to save in a String the time zone of the calendar and if for some reason is not possible to put the value from the calendar is set the UTC time zone.

**dayBefore**

This field is to set how many days before will start the synchronization.

**dayAfter**

This field is to set how many days after will start the synchronization.

**log**

This field is where the logger provided by the Funambol API will be set. It will allow setting in the logger all the steps helping very much in error detection. Than is possible to check the log inside the *logs* folder in the installation or in the application.

There are a couple of fields that will be used to set the URL feed to the calendars and to the events.

Now there is the methods that will allow the interaction with the Google calendar:

**login**

In this method is instantiate the service, to make this service valid we have to authenticate the user using the method *setUserCredentials* provided by the Google API. With the user authenticated we can access to the Google calendar of this user, if there was some problem in the authentication, user name invalid or password invalid, it throws *AuthenticationException* and it will be visible in the log. In these methods is set the URL feed of the calendar of this user.

**getAllEvents**

This method is to put all the events of the user into the field *listOfEvents.*

Using the service object it gets the events feed that allow to access to the event information, than all the entries are converted to *GoogleCalendarModel* with the method GoogleEvent2FoundationEvent, after this is ready to be inserted into the *listOfEvents*.

**getEventsByDateRangeQuery**

This method is to put all events of a specified date/time range into the field *listOfEvents*.

The implementation is quite similar to the previous method, the difference is when is getting the events feed, it gets with a query (*CalendarQuery*), this query have associated the date/time range of the information to get. After get the events feed it is the same as the previous method.

**getCalendarEvents**

This method is used to set in the field *listOfEvents* all the events of the authenticated user.

It is split in two parts, first part is if the parameters *dayBefore* and *dayAfter* are equal to zero, when this happen it will get all the events of the calendar calling the method *getAllEvents* explained before; second part when *dayBefore* and *dayAfter* have a value, this two parameters are used to get just a part of the calendar instead of getting all the calendar using the method *getEventsByDateRangeQuery* explained before too.

**createCalendarList**

This method is used o set the time zone of the calendar and to create the list of events.

To set the time zone it check what is the time zone of the first calendar and if there is some problem with the time zone of the calendar is set the UTC time zone, to create the list of events the method *getCalendarEvents* is called.

**removeCalendar**

This method is to remove the event that has correspondence with the id passed in the argument id.

It will load the Event associated with the id and then is deleted using the method delete, provided by the Google API.

**removeCalendars**

This method is to remove from the Google calendar every event that is in the argument *ArrayList* ids.

The *ArrayList* ids only have the ids of the events, so before start removing the events from the calendar they are put into an auxiliary *ArrayList* that will have all the objects *CalendarEventEntry* to be removed.

To remove the event is called the method *deleteEvents*, this method makes a batch request to delete all the events in the given list. It is a batch request instead of a simple delete, like the previous method, because this method can delete many events and that can bring some concurrency issues. If any of the operations fails, the errors returned from the server are displayed.

**updateCalendar**

This method is to update the event that is in the argument calendar type of *GoogleCalendarModel* into the Google calendar.

Is a long method but not complicated, it simply takes the information from the object calendar and prepare it to be updated in the Google calendar using the method update from the object service. In this method there are three important phases, first it take from the cache the event entry associated with the calendar to be updated, than sets all the properties in the entry and finally is updated into the Google calendar.

**insertCalendar**

This method is to insert the event that is in the argument calendar type of *GoogleCalendarModel* into the Google calendar.

The logic of this method is quite similar to the previous one, but instead of update it creates a new event in the Google calendar.

## Converters

**getDayOfWeekMaskFunambolToGoogle**

This method returns the conversion from the Funambol day of week type (represented in a *short int*) to the Google day of week type (represented in a *String*).

The day of week in Funambol is represented in *short int* and the day of week in Google is represented in *String*:

| short | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
|---|---|---|---|---|---|---|---|
| Funambol | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| Google | SU | MO | TU | WE | TH | FR | SA |

An example, if the number is 80 the return of this method will be “SU, TU”.

**convertFunambolTime2GoogleTime**

The Funambol time in the events is different than the Google time and this method is used to convert one type in to the other.

**GoogleEvent2CalendarModelEvent**

This method is used to convert a *GoogleEventEntry* into a *GoogleCalendarModel* object.

This conversion is made step by step, the information is taken out of the object *event* and converted and set into a object *GoogleCalendarModel*, that will be return at the end.


*Figure 5 - Information about GoogleCalendarDAO*

### 5.1.4. GoogleCalendarManager

This class makes a bridge between *Framework* and the S*yncsource*, is with this class that in the plug-in module we can access to the information in Google calendar.

This class extends from the *GoogleContactManager* because we can use some fields and methods to simplify the code:

**db**

This field is used to access to the Funambol database, is type of *DBAccess* that is in the Funambol API.

**writeChangesToDB**

This method is used to update all the changes in the database.

In the arguments of the method it receives a list of *SyncItems* to update, the source *URI* and the principal *ID* to access to the database, the update is made using the method of the *DBAccess* classe, *updateLocalItems*.

In this class there is some important fields:

**newItems**, **updatedItems**, **deletedItems**

These fields are type of *ArrayList<String>* and they save the ids of the new, updated and deleted items, all the changes will be saved in these lists.

**localItems**

This field is used as a cache of the database, is to avoid access directly to the database because is much slower.

**dao**

This field is the data access object.

Some of the methods in this class are a simple call of the data access object methods, this methods are the login, *getAllItemsKeys*, *getCalendarById*, *removeCalendars*, *removeCalendar*, *updateCalendar*, *insertItem.*

## Other methods

### loadItemsFromDb

This method set in the local cache, *localItems*, the items in the Funambol database, using the *db* field of the *GoogleContactManager* explained before.

### createNUDItemsList

In this method is where is created and set the three list of changes, new, update and delete.

First is created an enumerator of the list with all events in Google calendar, than one by one are checked:

If the event is the local cache is a possible update, but is just put in the *updated* list if the md5 hash of the two events are different;

If the event is not in the local cache is because is a new event and is added to the list *new*.

If there is some event that is in the local cache and is not in the Google calendar events it means that this event was deleted so is added to the *deleted* list.

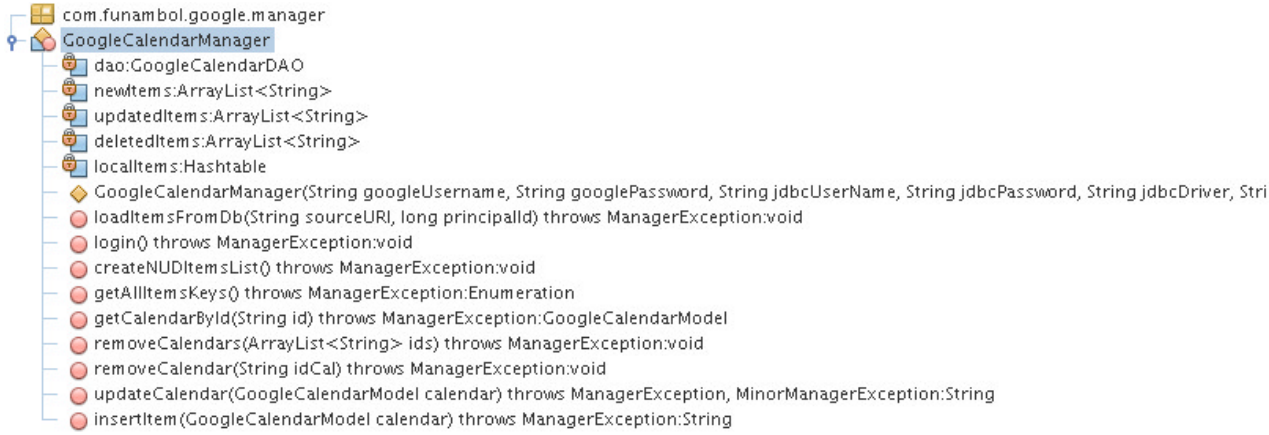


*Figure 6 - Information about GoogleCalendarManager*

## 5.2. Google Calendar SyncSource

This class is used to make the synchronization between the Funambol server and the Google calendar.

The *GoogleCalendarSynSource* is an extension of *GoogleContactsSyncSource* and is this one who implements the interface *SyncSource* of the Funambol. [3]

The implementation of some methods is quite the same, because the manager class who was explained before is the underground worker, the logic of the sync is quite the same.

### 5.2.1. Fields

**since** and **until**

These two fields are used to set a range of synchronization instead of all calendar synchronization.

**manager**

This field contains an instantiation of the *GoogleCalendarManager* class.

**calendarItemInfo2Store**

List used to set the modification when they are made in the side of Funambol server.

### 5.2.2. SyncSource interface

The methods of the *SyncSource* interface are what the Funambol engine will call to make the synchronization. [3][4][5]

**beginSync**

This is the first *SyncSource* method that the sync engine calls and it is used to specify who is going to synchronize and which type of synchronization is requested.

In this case the Google calendar manager will be prepared to do the synchronization.

In this method is where is made the login, is load the items from the Funambol database and is created the NUD (new, updated and deleted) lists and set in the field calendarItemInfo2Store all the events in Google calendar.

**commitSync**

Called to commit the changes applied during the synchronization session. Is used the field calendarItemInfo2Store to write the changes in the local data base using the method *writeChangesToDB* f the manager.

**getAllSyncItems**

Called to retrieve all the *SyncItems*, first it takes the keys of all items, and then adds to an *ArrayList* of *SyncItems* the *SyncItem* correspondent to the given key. The content of this *ArrayList* will be returned.

**getDeletedSyncItems**

Called to retrieve the deleted *SyncItems.* It is done using the manager to get the keys of all deleted items and then one by one create a list of *SyncItems* marking them as deleted.

```
returnList.add(new SyncItemImpl(this, items[i], SyncItemState.DELETED));
```

**getNewSyncItems**

Called to retrieve the new *SyncItems* and it is done using the manager to get the keys of all new items and then one by one create a list of *SyncItems* with the correspondent new item.

```
returnList.add(getSynciTemById(items[i]));
```

**getUpdatedSyncItems**

Called to retrieve the updated *SyncItems* and it is done using the manager to get the keys of all new items and then one by one create a list of *SyncItems* with the correspondent items to be updated.

```
returnList.add(getSynciTemById(items[i]));
```

**removeSyncItem**

Called to remove the given item and this logic is made in the manager side calling the method *removeCalendar*.

**setSyncItem**

Called to insert or update the given item.

If is to update it will call the auxiliary method *updateSyncItem,* this method will convert the *SyncItem* into a *GoogleCalendarModel* by calling the method *convertSyncItemToCalendar* (that will be explained later) after this it call the method *updateSyncItem* of the manager and then set the *timestamp* on the *hashtable calendarItemInfo2Store*;

If is a new item first it converts the calendar (*SIF* or *cal*) into a *GoogleCalendarModel* so like this it can call method *insertItem* of the manager, then it can set the id of this calendar and put it into the *hashtable calendarItemInfo2store* with a new *timestamp;*

Finally it returns the calendar has a *syncItem*.

### 5.2.3. Converters

**calendar2WebCalendar**

This method will convert Funambol calendar type (SIF) into a cal calendar type used by Google, the conversion is made by a method that the structure Funambol provides calendar2vcalendar of the class *VcalendarConverter*.

**webCalendar2Calendar**

This method is to convert a cal calendar into a *GoogleCalendarModel.*

First is converted the string *text* that have the content of the calendar into bytes, than is created a parser provided with the engine Funambol, after the parse the calendar is now a *vCalendar* type so now it can be converted with the Funambol method vcalendar2calendar that returns a *Calendar* type, now having the calendar in *Calendar* type it can be build a object type of *GoogleCalendarModel*.

**sif2Calendar**

This method is to convert a *SIF* calendar into a *GoogleCalendarModel*, the logic is the same as the previous conversion, but instead of a *vcalendar* is a *SIF* calendar.

**calendar2sif**

This method is to convert a calendar type of *Calendar* into a *SIF* calendar, the logic of this method is quite the same of the first converter, with the difference that it uses the Funambol object *CalendarToSIFE* to make the conversion.

### 5.2.4. Other methods

**getSyncItem**

This method is used to convert a calendar into a *SyncItem*, first is created a *SyncItem* and then is check what will be the conversion, converting to sif or to cal, this conversion return a String that have the content and the binary code of this content have to be set in a proprety:

```
syncItem.setProperty(
 new SyncItemProperty(SyncItem.PROPERTY_BINARY_CONTENT,
                                    content.getBytes())
);
```

**getSynciTemById**

This method only calls the method *getCalendarById* of the manager, getting the calendar and then converting it to a *SyncItem* that will be returned.

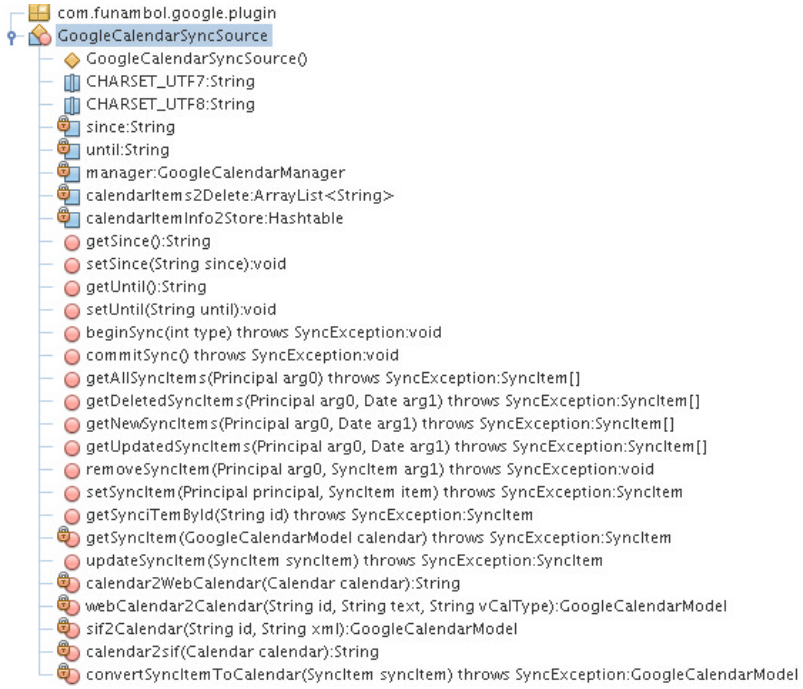


*Figure 7 - Information about GoogleCalendarSyncSource*

# 6. Building and running solution

The VMware image has everything set to build and run the solution. The user name is *fabio* and the password is *funambol*.

In this image you have access to several things, the forge of the Funambol is in the folder */home/fabio/v65*, the Google API is in the folder */home/fabio/project/gdata*, the installation of the Funambol is the folder */home/fabio/project/funambol*.

## 6.1. Building

Open the Net Beans and if the project is not already open the project that is the folder */home/fabio/project/funambol-gmail*.

All the module of the projects are in the left where is easy to see every file.

To build just make right click in Funambol Google Project, and make build.

## 6.2. Running

I created a file named iproject.sh that has all the function that we need to run and install the project.

In the terminal if you write:

`sh iproject.sh install` - it will install the plug-in.

`sh iproject.sh start` – it starts the Funambol server.

`sh iproject.sh stop` - it stops the Funambol server.

`sh iproject.sh plugin` – it starts the plug-in.

Than is just put the login and password of the Gmail and click in synchronize. I created an e-mail in the Gmail just for the tests, the user name is *funambol.gcalendarsync@gmail.com* and the password is *funambol*.

# 7. Conclusion

This project evolved many knowledge that I earn in this last three years, but more important than that it gave me other knowledge. It was with this project that I had my first contact with Linux, virtual machines and the open source community.

In the beginning I had many troubles to get use to the Linux, now I feel that I can easily work in Linux.

The virtual machine is very useful but quite simple to use, the only problem that I had was to configure the internet in the beginning.

The open source community is much more that I thought, there are people that are always there if we need help, I get some doubts about the Funambol server and they helped me a lot.

In the future this project can be improved to be compatible with other types of calendar, for example *ical*. But in a general way the project had a great success, all the objectives that were schedule in the beginning was reach at the end.

A conclusion that I can make about the Funambol is that is a great tool, because there is hundreds of compatible cell phones and the community is working to get more and more plug-in and is open source!

# 8. References and Resources


**Funambol**

[1] https://wiki.objectweb.org/sync4j/
[2] http://www.funambol.com/
[3] Funambol DS Server Architecture and Design Document
[4] Funambol ds server developer guide
[5] Funambol ds server module development tutorial

**Google Calendar API**

[6] http://code.google.com/apis/calendar/
[7] http://code.google.com/apis/gdata/javadoc/

**RFC 2445**

[8] http://code.google.com/p/Google-rfc-2445/
[9] http://tools.ietf.org/html/rfc2445
[10]http://en.wikipedia.org/wiki/ICalendar

**Maven**

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

**Vmware**

[12]http://www.vmware.com/

**Ubuntu**

[13]http://www.ubuntu.com/

**Others**

[14]http://www.google.com