Synchronization with External Task Systems in Octotask Application

Denis Laure, Yury Krupin, Alexander Abdulloev, Ilya Paramonov, Andrey Vasilev
Yaroslavl State University
Yaroslavl, Russia
{den.a.laure, krupiny, abdulloev.ivt, ivparamonov, vamonster}@gmail.com

Abstract

Octotask is a cross-platform task collector and manager, designed according to “Getting Things Done” style of time management. Its most important feature is synchronization with different task sources.

In this paper we discuss main ideas and algorithm of synchronization with multiple task sources for Octotask application. We define the external system interface, which allows to support different external systems in Octotask application.

Index Terms: GTD, synchronization, organizer, mobile device.

I. Introduction

“Getting Things Done” (GTD) is one of the most popular methodologies of personal time management. Its main goal is to increase personal productivity by unloading all the tasks from a person’s mind into an external system (GTD system) and using a special procedure to cope with them. All the tasks and task-related information from different sources are initially placed into the inbox folder. Afterwards contents of this folder is sorted according to the special procedure [1].

There are many GTD-compliant applications available. At the moment of writing, the website [2] provides information about 162 applications of such type. But this software does not settle one problem. We have many task sources from the computer world, such as issue tracking systems, e-mail, instant messaging services. When following the methodology, we should manually take or extract tasks from each of these sources and place them to the GTD system. It looks usual when we are using paper-based tools, but when our task sources already accessible in a digital form, it is desirable to place a burden of task collection to software.

The latter point was an impact for development of the Octotask project — GTD-style task collector and manager for mobile devices. Its main use case is the following. The user has multiple task sources (calendar of the mobile device, external todo lists, bug trackers, SMS and so on) and would like to have a unified overview of the tasks originated from all these sources. The Octotask application gathers the tasks from external systems automatically, organizes them in a local task storage accessible and alterable at any moment, and provide a way of synchronization between the local storage and all the external systems where the tasks originated from.

Such a multi-source task management requires to treat task synchronization in a different way than in common GTD managers. Instead of regular synchronization between the on-line service and the mobile client, designed for that service, which is usual nowadays, we need a common algorithm of synchronization, allowing arbitrary external system to be supported by means of some kind of common interface.

In this paper we present both the synchronization algoritm for transfer of differences between the local task storage and custom external system, and define the interface that has to be used to make such a transfer possible for an arbitrary external system.

The rest of this paper is organized as following. In section II we make an overview of the existing GTD organizers, show their strengths and weaknesses, and summarize that synchronization facilities in the existing software do not cover the use case we consider. Section III contains a description of main elements of our domain, namely Task, Context and Project entities. In section IV we define what is an external system and elaborate the interface of such a system. Section V describes the task synchronization algorithm we have developed for Octotask application. In conclusion we report about implementation of synchronization algorithm in Octotask and focus on future proposals for our application.

II. Overview of Existing Solutions

For now there are many organizers that helps to follow the GTD method more efficiently. All of them can be subdivided into two groups: services (Tracks, ToDoIst, Toodledo, etc.) and applications (EasyTask Manager, LeaderTask, iGTD, etc.). The applications of the both groups usually have almost the same functionality. It may include flexible task organization based on different principles (e.g. tree-like task structures or folders); additional information/files attachments; task filtering; notifications about tasks via email, SMS, Twitter, RSS or with the use of alarms; teamwork with task lists, allowing delegation of the tasks to teammates or co-workers.

here are two main approaches for such tools to take into account peculiarities of mobile devices. Firstly, almost every service provides some kind of mobile version, which supports the use of the service from mobile devices. Such versions are especially designed for small screen resolutions. Secondly, for some of the services there is specially developed client software. Both approaches have strengths and weaknesses. Strength of the mobile version approach is that there is no need to install any application to use the organizer. Another advantage is that such a version is usually quite suitable for each mobile device. The most significant weakness of this approach is that it depends on the Internet connection, so the absence of the latter means loss of possibility to work with the person's tasks at any moment.

Mobile clients for services usually do not require permanent internet connection. It is possible to manage the tasks offline and synchronize them with the service afterwards. But often people have to pay for such mobile clients (for example, Toodledo client for iOS [3]).

Both mobile clients and mobile versions of the services have a significant drawback: they are designed only for the one service, so if the user works with two or more task services, he cannot use all of them without neccessity to switch between different clients, which prevents the user from having the unified overview of all his/her tasks.

There are also standalone applications for mobile platforms. They allow to organize user's tasks directly on his/her mobile device and do not depend on any services. Usually such applications provide synchronization with only one (much less with only two) services and maybe with internal organizers and calendars of mobile devices. Good example of such an application is OrgYou S60 for Symbian [4]. It has rather simple UI and includes synchronization with the standard Symbian calendar and Outlook. Also OrgYou can save task list as an XML-file. One more example of mobile application is Pocket Informant. It is GTD style task manager designed for iOS, Android and Blackberry platforms [5]. It has very flexible interface, that allows to configure application as you like. This application can synchronize tasks with Google Calendar and Toodledo. Another good application for iOS platform is Action Lists [6]. It is

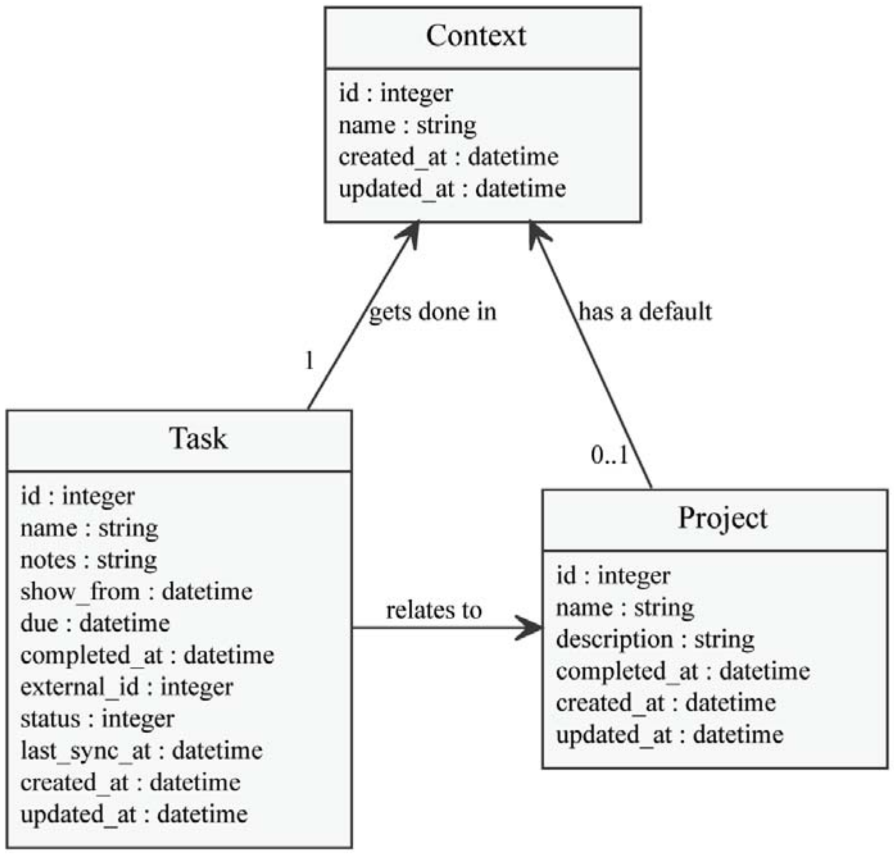

Fig. 1. Domain model diagram

rather a simple application especially designed to follow GTD methodology. It also allows to synchronize tasks with Toodledo service. So mobile applications are effective only when user uses one or two task sources. But, if he have many task sources (calendar, organizer, bugtracker etc.), there is no way for him/her to use mobile device as the general storage for all his tasks.

From this overview we can see that synchronization functionality in existing application is rather sparse, and there are no applications targeted at providing a common way for multi-source task management.

III. Domain model

The Octotask domain model is based upon the GTD methodology. It includes three entities: Task, Context, and Project. These entities relationships between them are depicted in fig. 1.

Task is the main entity of the domain model. It corresponds to the action which the person should perform. The “name” and “context” attributes are required, all the other attributes are optional. Notes (or description) attribute allows user to provide extra text information about the task. In order to make the task visible in the task list only from specific date we added “show from” attribute. The “due” attribute holds the due date for task. When the task is marked as done, the “completed at” attribute is filled with the current date and time. Task model also has some additional attributes required for synchronization process described below in section V.

Context is the second entity of the domain model. The meaning of Context in Octotask is the same as the context term in GTD methodology. Context is a some place or special conditions required to get the task done. “Home”, “office” and “free time” are most popular context examples. Each task refers to a context, and this link is mandatory.

The third entity is Project. Project is considered as an objective that takes more than one step to accomplish. The main properties of project are “name” and “description”. For each task the user can specify a project that task belongs to, but it is not mandatory, so it is possible to have a project-independent task.

Project has optional “default context” attribute. If default context is specified, it allows to automatically set context of the task associated with the project. Though, it is still possible to add tasks belonging to the different context.

IV. External System Interface

External system is some software task source, which can provide something that could be transformed into a task notable for the user. Basically, such a system also has to provide a way to retrieve the corresponding information from it via some kind of public API.

We subdivide all external systems into two classes: event-based and task-based systems. Examples of the systems from the first one are SMS, calls, emails. Information acquired by such a system, can be considered as an event, which has to be transformed into the task in Octotask. Such systems do not require any feedback about the task details or status to be transferred back to the external system. For example, email from the internet provider, informing about monthly payment, is an example of the event which causes ‘Pay the internet bill’ task.

Online GTD services, calendars and bug trackers are examples of task-based external systems. As a matter of fact, content of these systems is already represented in a form of tasks (for example, issues in a bug tracker can be easily considered in this way). So it is quite appropriate to talk about some kind of synchronization between the tasks in the external system and corresponding tasks in the Octotask application.

For example, the task from organizer about project deadline converts into task in mobile application. And any changes (like changing due time) applied to the task in application reflect on the task from external system. In a similar way, addition of a new task into the project, originated from the external system via the application UI leads to appearance of this task in the external system.

Octotask interacts with an external system via external system interface. Interface declares the most common operations for receiving and remote management of external system’s tasks. This interface includes the following operations:

1) Retrieval of active tasks from the external system
2) Modification of the task in the external system
3) Addition of the task to the external system
4) Removal of the task from the external system
5) Retrieval of one task by its id

The last three operations are mandatory for task-based systems. Event-based systems do not have to implement these functions as there is no feedback needed for this class of systems.

To allow support for a particular external system in Octotask we have to do the following:

- Provide an interpretation of how the concepts in the domain of the external system maps to the Octotask domain.

- Implement the external system interface for our particular system according to the mapping above.

V. Synchronization

In this section we describe task synchronization process between the local task storage and a task-based external system. The idea behind synchronization is to set the most recent values for attributes to the both local and remote representations of the task.

For each external system synchronization process starts from active tasks retrieval (operation 1 of external system interface, see sec. IV). Then the tasks, retrieved from the external system, and the corresponding tasks in the local storage are put in sync according to the procedure described in algorithm 1. The input of the procedure consists in two sets I and E defined in the header of the algorithm. Below we refer to the tasks from these sets as internal and external tasks respectively.

In order to determine the direction of task attribute transfer, each internal task has a status. There are the following statuses defined:

- “Synchronized means that the internal task was successfully synchronized with the corresponding external task and was not internally modified since that time.
- “Locally changed means that the internal task was either added to the local task storage or modified since the last synchronization.
- “Locally removed means that the internal task was removed from local storage since last synchronization.

There are also some service attributes which are necessary for the synchronization procedure. The first attribute is “updated at timestamp. It relates to both internal and external tasks and contains the time of the most recent modification of the task. The other attribute is “last sync at attribute of internal task. We update this timestamp on retrieval of information about the particular task from the external system or when we update external task.

According to synchronization algorithm, for each internal task we try to find the corresponding external task. On success we determine, which of the tasks from the pair has been changed since the last synchronization. The local task is considered to be modified when its status is not “synchronized. We treat the external task modified when its “updated at timestamp is latter than “last sync at timestamp. Thus, we have four possible combinations, and each of them implies an appropriate action to be made upon the tasks to make them synchronized.

If only the internal task was modified, its status is checked in order to determine the character of the change. If the task was removed from the local task storage, then it will be removed in external system (operation 4). If the internal task was modified (for example, renamed or due time was changed), the corresponding changes will be applied to the external task (operation 2). If only the external task was modified, then corresponding internal task is updated accordingly. If both the internal and external tasks were modified we need to ask user to resolve the conflict.

After the search is done, there could be tasks without a pair in both sets of tasks. if the pair was not found for the internal task, it could mean one of the three thing:

- the task was closed in the external system;
- it was removed from there;
- the task was added in the local storage.

In the first case the closed external task is retrieved (operation 5) and the pair of internal and external task is compared according to previously described algorithm.

Algorithm 1 Task synchronization procedure

Input: I — set of internal tasks, which belong to all projects, related to the external system S, except for completed in-sync tasks; E — set of all active tasks retrieved from the external system S.

```
for $int \in I$ do
  if exists $ext \in E$, corresponding to $int$ then
    if in $(int, ext)$ only $ext$ was modified since the last synchronization then
      modify $int$ in the local storage according to $ext$
    else if in $(int, ext)$ only $int$ was modified since the last synchronization then
      update $ext$ on the server according to $int$
      $int.status \leftarrow synchronized$
    else if both $int$ and $ext$ were modified since the last synchronization then
      resolve the conflict
    end if
    remove $ext$ from $E$
  else
    if $int$ has $external\_id$ set then
      retrieve the task with $id = int.external\_id$ from $S$ as $ext$
      if $ext$ was deleted from $S$ then
        if $int$ locally changed then
          resolve the conflict
        else
          remove $int$ from the local storage
        end if
      else
        apply the rules above
      end if
    else
      add new task to $S$
      fill in $external\_id$ in $int$
    end if
  end if
end for
for $ext \in E$ do
  if The local storage contains the task $int$, corresponding to $ext$ then
    apply the changes to $int$
    $int.status \leftarrow synchronized$
  else
    add new task, corresponding to $ext$, to the local storage
  end if
end for
```

In the second case local task status is checked. If the task was also marked as removed in the local storage, it will be deleted from the storage. But if the local task was modified, the user must resolve the conflict. In the third case the new task will be created in the external system (operation 3) and associated with the local task. To distinguish these situations the "external id" task attribute is used. If it is set, this means that local task was associated before and corresponding external task was removed or closed. And if attribute is not set, the local task was added in the local storage.

There could be external tasks for which there was no internal task found in set I. For each of them we look for any internal task in the local storage which is associated with the external task. If the corresponding task was found, the local task is synchronized with the external task. Elsewhere the new local task is created for the external task and associated with it.

At last, we should consider the periodicity of the synchronization process. On the one hand user should always work with the most actual tasks. On the other hand synchronization with remote task systems requires connection to the internet, which consumes power and network traffic.

We decided to start the synchronization process in the following cases. Firstly, when user changes the task, which is associated with the external project, because we need to apply corresponding change to the task in the external system as soon as possible. Secondly, when the device is connected to the internet, we periodically retrieve all changes made in the external systems since the last synchronization. Thirdly, user can optionally start the process manually.

VI. Conclusion

In this paper we proposed the mechanism of task synchronization for the task manager and multiple external task systems. Our approach allows to implement use case of multi-source task managent when the tasks in the local storage of the mobile application are synchronized with arbitrary external systems via specially defined external system interface. The developed algoritm of synchronization does not depend on concrete external systems.

This approach was successfully implemented in Octotask application, which for now supports synchronization with Redmine project management system and Google Tasks service.

Octotask is an Open source application available for Symbian and Harmattan platforms. Octotask homepage is available at *https://yar.fruct.org/projects/octotask*.

One of the most important direction of the future development of the project is elaboration of some kind of plug-in system to simplify support for arbitrary external systems in the application.

Improvement of synchronization algorithm would include solution of concurrency issues emerging when contents of the local or remote task storage is modified during synchronization.

We also plan to add support for synchronization with the built-in Symbian and Harmattan calendars, different online organizers having open API (Toodledo, Nozbe). Also several event-based systems would be covered. For example, SMS and missed calls would be easily transformed into tasks in Octotask.

References

[1] D. Allen. "Getting Things Done: The Art of Stress-Free Productivity," *Penguin Books*, 2001.
[2] GTD Software Comparison — 162 Researched Apps. http://www.priacta.com/Articles/Comparison_of_GTD_Software.php
[3] Toodledo client for iOS. http://www.toodledo.com/info/iphone.php
[4] OrgYou S60. http://www.orgyou.eu/
[5] Pocket Informant. http://www.pocketinformant.com/index.php
[6] Action Lists. http://software.dazeend.org/action_lists/index.html