VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

COOPERATIVE PROJECT PLANNING IN GETTING THINGS GNOME

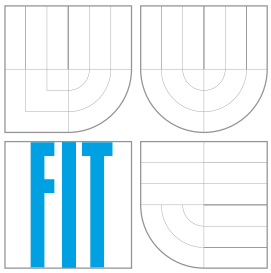
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. IZIDOR MATUŠOV

BRNO 2013

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

KOOPERATIVNÍ PLÁNOVÁNÍ PROJEKTŮ V APLIKACI GETTING THINGS GNOME

COOPERATIVE PROJECT PLANNING IN GETTING THINGS GNOME

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. IZIDOR MATUŠOV

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. ALEŠ SMRČKA, Ph.D.

BRNO 2013

Abstrakt

Tato práce pojednává o rozšíření pro aplikaci Getting Things GNOME, které ji umožní používat pro plánování a správu kooperativních projektů. Čtenář je seznámen se základními principy a vybranými metodami plánování a řízení projektů. V rámci této práce byla identifikovaná cílová skupina uživatelů a jejich potřeb, pro které bylo navrhnuto řešení. Práce obsahuje přehled rozšířeného uživatelského rozhraní pomocí náčrtků. Navrhnuté řešení bylo implementováno a otestováno.

Abstract

This work dicusses the extension for Getting Things GNOME which makes it possible to use it for planning and managment of cooperative projects. Reader is introduced to basics of project planning and project management and selected methods. The target audience and their needs for the extension were identified and their solutions were proposed. This work includes a walkthrough the extension of the user interface in the form of wireframes. The proposed extension was implemented and tested.

Klíčová slova

Getting Things Gnome, GTG plánování projektu, projektový management, kooperativní projekty, synchronizace, XMPP, PubSub

Keywords

Getting Things GNOME, GTG, project planning, project management, cooperative projects, synchronization, XMPP, PubSub

Citace

Izidor Matušov: Cooperative Project Planning in Getting Things GNOME, diplomová práce, Brno, FIT VUT v Brně, 2013

Cooperative Project Planning in Getting Things GNOME

Declaration

I declare that this work is my own account of my research and have been completed under the lead of Aleš Smrčka and Lionel Dricot.

.....
Izidor Matušov
May 22, 2013

Acknowledgements

Many Thanks to my thesis advisor Ales Smrčka and Lionel Dricot for their guidance, and priceless advices. Thanks to my family and friends who supported me during fulfilling this work. I would like to thank especially to Google Inc. and GNOME Foundation that part of this work could be done as Google Summer of Code project.

Contents

1 Introduction **4**

2 State of the Art **5**
2.1 Project Planning and Management 5
2.1.1 Three-point Estimation Method 6
2.1.2 Evidence Based Scheduling 6
2.1.3 Existing Solutions 6
2.2 Getting Things GNOME 7

3 Design of the System **10**
3.1 Target User Group 10
3.2 The Need of the Users 11
3.2.1 Sharing Tasks and Offline Mode 11
3.2.2 Management of a Team 12
3.2.3 Management and Assignment of Project Tasks 12
3.2.4 Estimations 12
3.3 Walkthrough of the Extension of the User Interface 14

4 Implementation of the System **21**
4.1 Tools Used for Implementation 21
4.2 Extension of GTG Infrastructure 21
4.2.1 Concept of Contacts 22
4.2.2 Consequences of Contacts Aggregation 22
4.2.3 Extending Tag Object 23
4.2.4 Extending Task Object 23
4.3 XMPP Synchronization 24
4.3.1 PubSub for XMPP Synchronization in GTG 24
4.3.2 Tools Chosen for Implementation 25
4.3.3 Overview of Implementation of XMPP Synchronization 26
4.3.4 Solving Synchronization Conflicts 26
4.3.5 Preventing Self-Changed Loop 27
4.3.6 Storying Content of Tasks in Persistent Items 28
4.3.7 Maintaining Permission Model 29
4.3.8 Auto-Discovery of Relevant Nodes 30
4.3.9 Synchronization of Non-Tasks Data 30
4.4 Implementation of the Extension of the User Interface 31
4.4.1 Tag Editor 31
4.4.2 Task Editor 32

4.5 Merge Request 33

5 Usability testing **34**
5.1 Form of the Usability Testing 34
5.2 Tested Use Case 34
5.3 Observations 35
5.4 Discussion of Identified Issues 37

6 Conclusion **38**

A Installation and Configuring XMPP Server **42**

List of Figures

2.1 Screenshots of the user interface of GTG. Left: the browser of tasks, right: an editor of a task which can be opened by a double click on the task in the browser. 8
2.2 Architecture of GTG 8
2.3 A configuration dialog of a synchronization service 9

3.1 Tag edit can be accessed from the main window (left) by a right click on a tag which should be shared and selecting Edit tag item (right). 15
3.2 If there is no synchronization service with the support of sharing, the user is encouraged to add a new one. 16
3.3 The content of the tab *Sharing* when a synchronization service with sharing supports is activated (left). Confirmation dialog to remove a contact „Joe“(right). 16
3.4 Tab *Calendar* can set the amount of hours working per workday. The user can list the exceptions like holidays or vacation. 17
3.5 Tab *Reports* provide the overview of the activity given the time range. . . . 17
3.6 A task without an estimation nags the user to enter it. 18
3.7 A special bar indicates time tracking. 19
3.8 Report on the task. Work entries which could be edit on the left, the overview of estimations on the right. 19
3.9 Dialog for solving conflicts when the user was offline 20

4.1 PubSub architecture adapted for needs of XMPP Synchronization in GTG . 25
4.2 Finite state machine that facilitates the implementation of offline mode support 27
4.3 Sequence of passing changes that leads into a loop 28

5.1 Two standard screens used to add a new service and its configuration. . . . 36

Chapter 1

Introduction

Getting Things GNOME (GTG) is a todo list and task manager for GNOME desktop environment. GTG has features the user needs to manage her own tasks with some rare features like WorkView or unlimited level of subtasks. The current world with the accelerating changes require people to work together in teams. Although the user as an individual can manage her tasks in GTG, the cooperative project can not be managed in the same way.

However, a cooperative project is not so different than a task list of an individual. The project is just a big task that can be split into many small, manageable subtasks which are regular tasks and have attributes like due dates. The project can be changed and reorganized in the same way as tasks of individual. There is need for additional concepts like sharing the project and its subtasks with other members of the team, management of the team, assigning tasks to a member of the team and reporting.

Existing solutions like Microsoft Project or TaskJuggler requires the manager of the team to describe the project and create a similar subprojects in the application. She has to update the state of the project herself and only she has the overview of the state of the project. The task based approach is more transparent where every member can update and see the state of the project.

Solutions based on groupware like BaseCamp and Redmine provide the transparency but lack the focus on the project planning and the completion under constraints. The task based approach can use established methods in the field and provide those tools to the users to facilitate the project planning and management.

Chapter 2 discusses the state of the art in the project management including couple of estimation methods. Existing solutions in the field are presented. In Chapter 2.2 GTG and its architecture is described. Chapter 3 is designated to the design of the extension to GTG. The target audience and their needs are presented. Feasible solutions for every of the needs is discussed. In Chapter 3.3, wireframes of the extended user interface are presented with brief comments. The implementation of the proposed extension is described in Chapter 4. The description of the changes to the GTG infrastructure is described in Chapter 4.2. Details of XMPP synchronization are listed in Chapter 4.3.1. In Chapter 4.4 the noteworthy decisions when implementing the user interface are mention. Chapter 5 presents the user testing of the implemented extension.

Chapter 2

State of the Art

This chapter introduces the reader to project management and planning concepts. Later the overview of Getting Things GNOME (GTG) is presented.

2.1 Project Planning and Management

By Project Planning and Management we can understand the process of accomplishing a goal given limited resources. Resource is a generic term describing input needed to complete the project, typically employees and material for the project. The manager of a project is a person who plans and later manage activities towards the goal and reach up to 2 of 3 qualities:

- high quality of work,
- short time needed to finish the work, and/or
- low prize of the work – amount of used resources.

In the first stage of the project, the manager has to analyze the situation, activities needed to complete the goal and available resources. To analyze activities and estimate needed time so-called Work Breakdown Structure is used. An activity is break down to more smaller activities. It is repeated until the activities are small enough to be easily manageable. The activities are estimated how much resources and time they need, dependencies among activities, e.g. the roof of the house can not be deployed until there are walls. To each activity needed resources are assigned, including employees working on it.

Critical path method is used to analyze the activities and find out how they should be scheduled to complete the project in the shortest time. Using more employees might speed up the execution of activities as more activities can be work on in parallel. Removing unnecessary activities decrease the needed time but also decrease the quality of work.

When the scheduling the activities is completed, the manager have a plan for the project, an estimated cost of the project and an estimated completion date of the project. The manager should predict possible risks that can occur and threat the completion of the project. She can do that by adding redundant resources, additional time or mark some activities as unnecessary which could be removed if needed.

If the plan gets approved, the next stage is the management of the project. The manager checks that activities are executed according to the schedule and keep the project on the track. If unpredicted activities are discovered, estimations were wrong, or resources are

lost like an employee leaves the company, the manager has to solve the situation. She can either add more resources, remove unneeded activities or postpone the completion date. More details can be found in [9].

2.1.1 Three-point Estimation Method

To make a realistic estimate how much time an activity takes is hard. Managers often overestimate or underestimate the complexity of an activity what makes realistic scheduling hard. Three-point Estimation Method addresses this issue by requiring a manager to enter 3 estimations:

- estimation in the best case,
- estimation in the most likely case, and
- estimation in the worst case.

One might see those 3 estimations as minimum, median and maximum of the virtual time estimation distribution. When scheduling, the estimation can be transformed to expected estimation using Monte Carlo simulation or as a weighted average

$$< estimation >= \frac{1}{6} < bestcase > + \frac{4}{6} < mostlikelycase > + \frac{1}{6} < worstcase >$$

when the most likely case happens in 80% of times, the best and the worst case happen in 10% of times each. More details on Three-point Estimation Method can be found in [5].

2.1.2 Evidence Based Scheduling

Evidence Based Scheduling [20] is a method proposed by Joel Spolsky and implemented in the bug tracking software called FogBugz. The tracked bugs are assigned to developers. The assigned developer make an estimation for the bug she is assigned to. As the developers work on their bugs, they track the time spent on the bug.

The manager of the project does not create any plan beforehand, just decide which bugs are going to be fixed. The software creates the plan itself based on the evidence how much time a developer spent on a bug versus the estimation made by the developer. If a developer usually underestimates or overestimates, the software learns that and learns by how much. Based on this evidence, the software estimates the completion date.

2.1.3 Existing Solutions

The flagship of project planning and management solutions is Microsoft Project [14] where the most of techniques mentioned above can be used. It is a proprietary solution and there exists many open sources clones, e.g. TaskJuggler [18]. The target audience is an individual manager that manages the project alone in the software what usually makes it non transparent to employees.

Another kind of solutions is groupware software. It is typically a web based system where people can create teams, share files, create tasks, and chat. However, the focus is not on the existing project planning and management methods outlined above. Examples are

BaseCamp from 37signals [3] or Trello from Fog Creek Software [19]. Open source solution is the bug tracking software Redmine [12].

SharePoint from Microsoft [15] seems to support cooperative project planning and management. However, this solution is proprietary.

2.2 Getting Things GNOME

Getting Things GNOME! (GTG) [2] is a todo list and task manager for GNOME desktop environment. The project was founded by two Belgians back in October 2008. with the first public release in March 2009 [7]. The original founders were Lionel Dricot and Bertrand Rousseau. Until the current GTG 0.3 released in November 2012 there 65 contributors to GTG. You can see the visualized history at [13].

The name of the application is a word play on a popular method Getting Things Done method [4]. Although GTG is inspired by the Getting Things Done method, it does not strongly require users to follow the method. The most important features are:

- the initial task set is a tutorial how to use GTG,
- a task can have start and due date, due date can be even a fuzzy value like now, soon or someday ,
- a task can have subtasks and subtasks can also have subtasks – unlimited level of subtasks,
- the hierarchy of tasks is represented as a directed acyclic graph, i.e. one subtask can have multiple parent tasks ,
- a task can be assigned tags which represents context or category of the task and is useful for quick filtering of tasks,
- a task is edited in richtext editor with a simple user interface ,
- the WorkView mode shows only tasks you can work on now, i.e. the start date of a task is passed, there are no active subtasks to be done,
- a search which can be stored as a special tag for future quick filtering, and
- extensibility via synchronization services or plugins.

The important feature of GTG is its extensibility. Its architecture as depicted in Figure 2.2 can be divided into 4 parts: the user interface, liblarch, plugins, synchronization services. Liblarch has been a standalone library since GTG 0.2.9. It is a library that can handle directed acyclic graphs and represent it in a tree, e.g. in a TreeWidget of GTG's browser. Liblarch can make it easy to filter tasks by filter functions and show the graph in multiple trees at the same time. The library was extracted from GTG code base in order to provide solution to other projects if needed.

Plugins were the first way how to extend GTG. Plugins engine was written by Paulo Cabido in summer 2009 [6]. A plugin can extend the user interface to provide features which might not be suitable for all users of GTG. Examples of such plugins are a plugin for the time tracking tool *Hamster* or a plugin *Do it Tomorrow*.

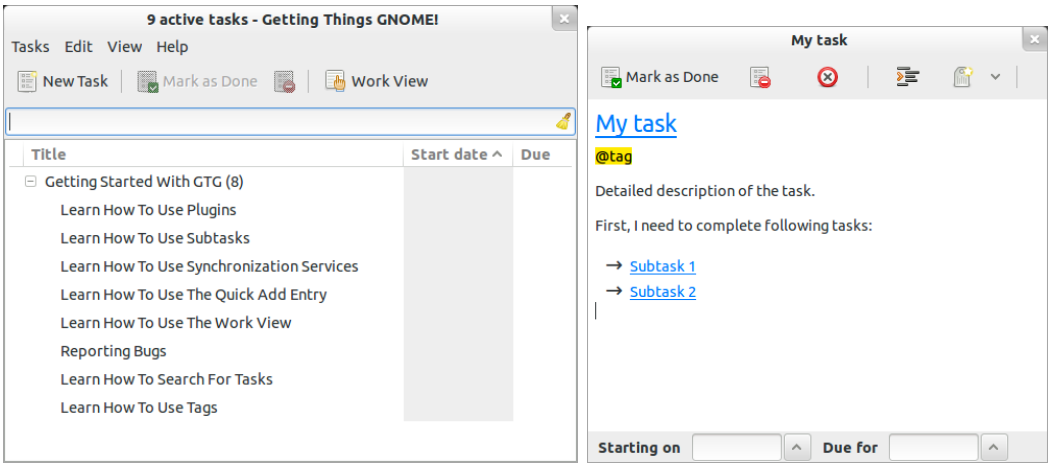


Figure 2.1: Screenshots of the user interface of GTG. Left: the browser of tasks, right: an editor of a task which can be opened by a double click on the task in the browser.

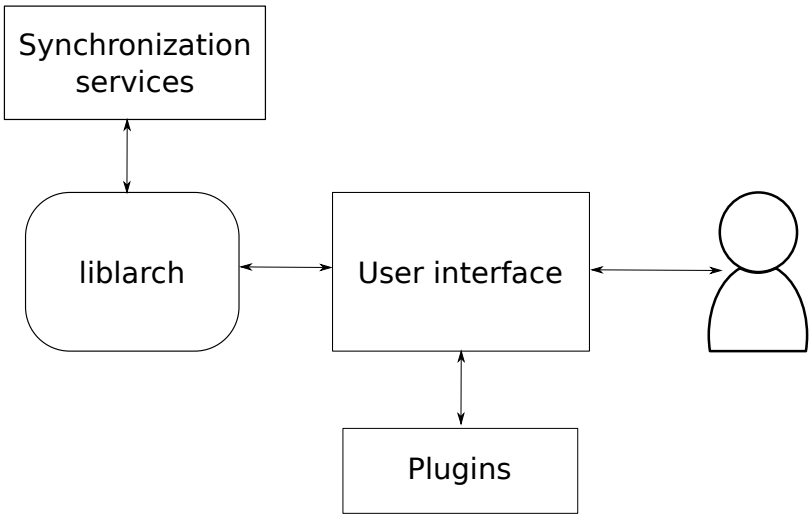


Figure 2.2: Architecture of GTG

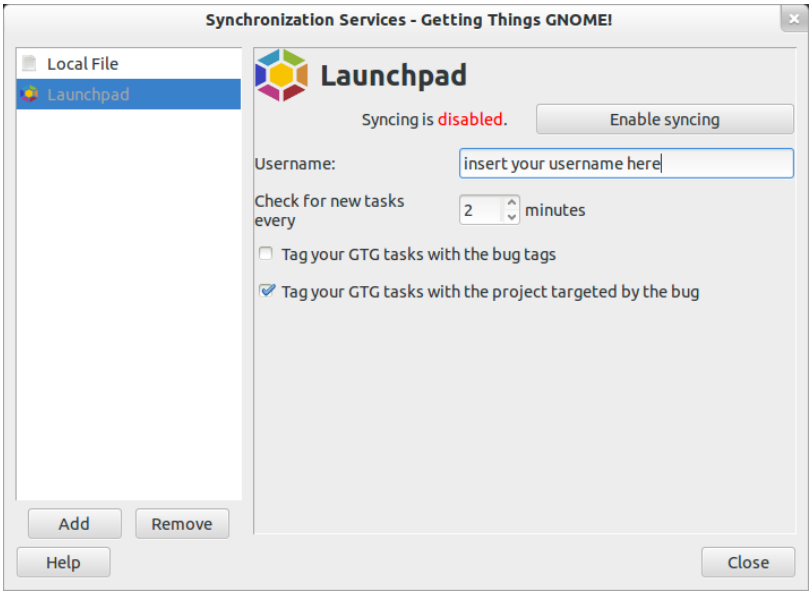


Figure 2.3: A configuration dialog of a synchronization service

Luca Invernizzi implemented synchronization services engine[1] back in summer 2010 [10]. Services allow the user to synchronize their tasks to a remote service like Remember The Milk every couple minutes or import bugs from an issue tracker like Launchpad. The engine provides way to run multiple services at the same time and imports and exports tasks directly from or to liblarch model of tasks. A synchronization service does not require to modify the user interface because the engine generate configuration dialogs based on its parameters. An example of such a dialog is depicted in Figure 2.3.

[1]The original name was backend engine. It was more term from a point of view of a programmer and in summer 2012 the community decided to rename it. More information at https://bugs.launchpad.net/gtg/+bug/962642

Chapter 3

Design of the System

This chapter proposes the design of the solution – the extension for GTG for cooperative project planning.

3.1 Target User Group

The solution is aimed at a company where people work in small teams. The trust is important part of the culture of the company that makes the work simple and transparent. The company owns or has rented its own IT infrastructure, e.g. a mail server, an XMPP server. Employees communicate via XMPP, either via direct messages or multi-user chat rooms, and they already have all co-workers in their contact list associated with their account.

The requirement of small teams implicates that a complicated solution for managing bigger amount of employees is not needed. As proposed in the Mythical Man Month [11], the overhead of the team of size n because of the communication within team is about $n * (n - 1)/2$. In other words, the complexity grows much quicker than the size of the team and smaller teams have the advantage of the smaller overhead. Although the proposal is aimed at teams of software developers, it could be applied to other areas of project management as well.

The company can be divided into multiple teams by projects or department. An employee can be a member of multiple teams.

Every member of the team is trusted to work towards the completion of the plan and has the access to all needed information. A member can see reports on the state of the project and take the action if needed, e.g. to notify their manager about the project slipping behind the schedule. A member can invite a new member or revoke the access to a former member who do not need it anymore.

The result is a simple and transparent solution without complicated permission model. Either an employee is a member of a team or is not. The purpose of the designed solution is to not stand in the way of employees.

The synchronizations is based on the server-client model where the full copy of data is stored on company's server and is synchronized to employees' computers. The server is backuped and in the case of a mistake or misuse of the trust, data can be restored.

The synchronization works as the extension of XMPP protocol what brings two major advantages:

- Identities for employees can be still managed on XMPP server, which could be linked

to an LDAP server or other infrastructure of the company. There is no additional burden for the system administrator.

- Employees already have all they co-workers in their contact list. It lowers the time needed for the setup.

3.2 The Need of the Users

When the users work on a cooperative project, there are several needs that need to be satisfied:

- sharing tasks with the members of a team,
- the management of a team,
- the management of project tasks,
- the assignment of tasks to co-workers,
- the estimation of the duration and the completion date of the project,
- reports of the state of the project, and
- offline mode.

The proposed solution does not solve the problem of sharing files like text documents or spreadsheets associated with a task. The users can store those files in a dedicated service and attach a link to the files in the description of the task.

3.2.1 Sharing Tasks and Offline Mode

The best way to extend GTG with sharing capabilities is a new synchronization service. The synchronization happens as an extension of XMPP protocol, namely XEP 0060: Publish-Subscribe [16]. When an instance of GTG is running and online, it subscribes to data on the XMPP server. On the behalf of the user, an instance of GTG publish modified data to the server and the server sends a notification to all subscribed instances in a form of an XMPP message.

When an user is offline, she still can work with her GTG. The situation can happen to her when she travels and the internet connection is not available. In such a case, the modifications are stored offline and synchronized when the server is available again. If more than one user modified the same portion of data, merge conflicts might occur as it is the case by distributed version control systems. None of the conflicts are solved by GTG itself and the user uploading changes is asked for her input: use the remote version, use her version or merge changes manually. An automated resolving of a subset of the conflicts is out of the scope of this work. The purpose of the synchronization service is also to provide meta-data like the contact list which represents user's co-workers and a list of teams which the user is a member of.

3.2.2 Management of a Team

The concept of tags in GTG is to mark tasks with the common context and thus it makes sense to represent a team as a tag. Tasks with the team tag are shared to co-workers. GTG has already a dialog for editing parameters of a tag called tag editor. It can be extended to manage settings and view reports related to the team.

3.2.3 Management and Assignment of Project Tasks

The project tasks can be handled in the same way as regular GTG tasks. Tasks have their parameters and dependencies between tasks can be represented in task - subtask relation. Tasks which need to be completed first are represented as subtasks. GTG does not provide support only to a tree of tasks but a directed acyclic graph of tasks. It means that a subtask can have more than one parent. It is possible to represent even more complex relations between tasks. The current model of a task in GTG does not have a concept of an assignee, an owner of the task and thus it is needed to extend it for this parameter.

3.2.4 Estimations

The duration represented as how many man-hours are needed and the completion date of the project represented as when the last task will be completed are two different views on the same parameter of the project. Which view is more important to the user depends on the goal of the user. A consulting firm prefers the duration; the price of the project is based on the count of man-hours multiplied by the hourly rate. On the other side, a product-based company is more interested in the completion date. Both views are important and should be easy to find in the user interface.

General estimations In order to estimate either the duration or the completion date, team members need to break down the big project task into many small tasks in the same way as it is by work breakdown structure. When finished, each task can be completed in the couple hours or in the couple days, otherwise tasks should be break down to even smaller tasks. Relations between tasks should be entered at this stage.

The next step, every team member estimates the effort needed to complete a task in man-hours or man-days. One man-day represents 8 man-hours. This work proposes a single estimation. In a future version, a more sophisticated estimation system like three points estimation method can be used if needed. The latter method requires the user to enter three estimation which are transformed into a single estimation as a weighted average.

The estimation for a task represents the effort needed to complete just the task itself and does not contain the effort needed for subtasks. It implies that a parent task can need a smaller effort than all its subtasks combined.

Although the proposed system can do initial estimations for the project, estimations updated on the progress are equally important. Although tasks can be small enough to track the progress, tasks running late are not easily discoverable. A possible solution is a parameter progress for a task. Unlike other alternatives, GTG does not support this concept. The user would need to update this parameter manually or she can experience so-called *90% completed* syndrome (The user can't properly estimate the duration of the task because she does not know all details of it; at every stage the task seems to be 90% completed). Another solution is to track the time spent on a task. The progress is the ratio

of time spent and time estimated. If the execution of a task is running late or no progress is made, it is easy to discover those tasks.

Time tracker A tracker of the time already spent on a task is needed. Painless time tracker is a challenge to design. The user needs to start the tracking when she starts working on a task and end it when she interrupts the work. The user must be disciplined and create the required habit. If the user forgets about the tracking, she has to enter data manually. It is easier for her as a human being to remember how many hours she spent on the task, than the start and the end of the work which is the machine approach.

This work proposes to create a time tracker built in GTG. Time spent on a task is tracked when the editor window of the task has been opened by the user who is assigned to the task. She begins the work in the moment she opens the window and finishes when the window is closed.

The idea behind the proposal is that the user can keep her notes like e-mail addresses, ideas, links to files in the description of the task. The effort connected with the tracking is controlled by a simple state of opened window. If the user forgets to open the window, she can edit an entry. It stores only the number of hours spent on a given day instead of the start and the end of the work. Finer details like the start and the end of work are not needed for the estimation anyway.

As a counter solution, there already is a plugin to track GTG tasks in the time tracker for GNOME desktop environment called *Hamster*. The advantages of the plugin are that it is already implemented and some users of GTG already use it. The disadvantages of the plugin are that it is hard to match entries in *Hamster* with GTG tasks if the user starts the task via *Hamster* directly and not the via the plugin. If the user works on multiple computers, there might be problems to keep the integrity among instances of GTG and instances of *Hamster*.

The built-in time tracker would provide a better user experience. The problems with the integrity would be solved through the new synchronization mechanism. Therefore the built-in time tracker is used.

Completion date estimation The estimation of the completion date is slightly harder than the estimation of the duration because users work only limited hours a day based on their contracts, usually 8 hours a workday. If the project is completed in 74 man-hours, the completion date for a single user is not in $74/24 = 3$ days but in $74/8 = 9.25$ standard workdays. Therefore every team member has its own calendar. The calendar consists of the number of hours the user works in the team on given day and list of exceptions when the user does not work at all, e.g. national holidays or vacation. An example of such a calendar is in Table 3.1. The owner of the calendar can work 8 hours Mondays, Tuesdays and Wednesdays, but only 4 hours Thursdays and Fridays. She does not work Saturdays and Sundays and has holidays on March 4 - 6.

The calendar allows to specify that an employee works only part time or that an employee works certain days as a member of one team and the rest of her work time as a member of another one.

When the completion date is estimated, all active tasks, their dependencies, the effort needed to completed and assigned persons to them are considered. If there is no person assigned to a task, any team member can be assigned to the task virtually for the purpose of the computation. When all tasks have assigned members, the critical path method is

Day	Hours
Monday	8
Tuesday	8
Wednesday	8
Thursday	4
Friday	4
Saturday	0
Sunday	0

Exceptions:
2013-03-04
2013-03-05
2013-03-06

Table 3.1: Example of a calendar

used to find the critical path and return the estimated end of the last task at the end of the critical path.

When the members of a team put down their estimation of the duration of a task, they estimate how much time the average member of the team needs to complete the task. By the definition, approximately half of the members completes the task in shorter time, the other half needs more time. This work proposes to adapt the approach used in Evidence Based Scheduling [20]. The machine learning based system learns about time spent on a task versus its average estimation for each member of the team. The gained knowledge is then used to compute the estimation that a given team member. For example, if a member usually needs only 75% of the estimated time and the task she is working on has the estimation 3 hours, she will probably complete the task in $2\frac{1}{4}$ hours. If her performance decreases, the system notices it and increases the ratio of needed time to estimated time gradually.

In Evidence Based Scheduling, only the person assigned to a task can put the estimation in order to prevent a manager playing the *tight schedule* trick; the person assigned to the task should complete the task in very short, often unrealistic time. In the proposed solution, a manager has only one vote and has therefore less importance. Members have a different level of expertise, different experiences and different estimations. Senior members might estimate better the possible problems a task can run into, junior members might use a brand new technique in order to solve a task in shorter time. The average estimation reduces big biases and provides a realistic estimation.

3.3 Walkthrough of the Extension of the User Interface

The setup of a cooperative project can be done via tag editor. It is an advanced feature, used only few times and thus control widgets are not accessible right from the main window of GTG. Tag editor dialog is extended by multiple tabs. The current settings like the name and the color of a tag are left as they are in tab *General*. Three new tabs are added:

- sharing – the management of members of team,

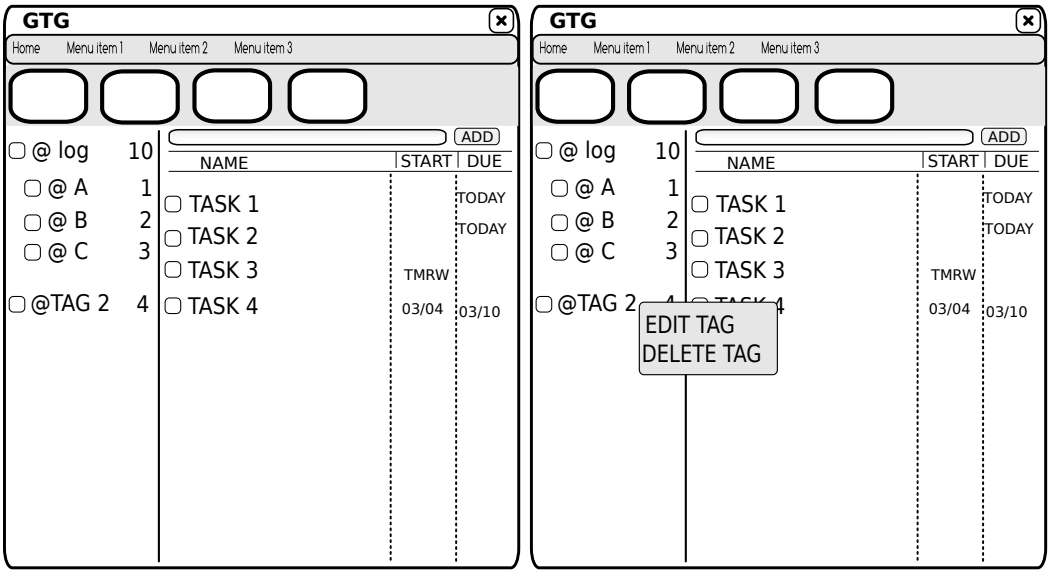


Figure 3.1: Tag edit can be accessed from the main window (left) by a right click on a tag which should be shared and selecting Edit tag item (right).

- calendar – the management of user's calendar for the given team, and
- reports – reports of the team.

When the user does not have any synchronization service supporting the sharing of tasks, only tabs General and Sharing are shown. The tab *Sharing* contains a message that the user has to add a new synchronization service which supports sharing and a button which opens the dialog for Synchronization services (Figure 2.3). There is a help button in the bottom right corner if the user would like to read more about cooperative projects. The wireframe of the dialog is depicted in Figure 3.2.

If there is at least one active synchronization service with the support of the sharing, then the tab *Sharing* contains a list of contacts the user can share tasks to (Figure 3.3). The list contains a checkbox, the avatar and the name of a contact. If the checkbox is checked, the contact is a member of the team. The avatar and the name are provided by the synchronization service. In the case of XMPP extension, the avatar is the XMPP avatar of the contact and the name is the nickname the user gave to the contact in her contact list. There is a special, first contact labeled *Me*. It represents the user herself. If the user want to remove from the team, she can uncheck *Me* contact. Removing a member of the team is potentially dangerous situation and must be confirmed in a modal dialog in the same way as deleting tasks in GTG or deleting files in a file manager.

Tab *Calendar* (Figure 3.4) consists of two widgets. The first widget represents amount of hours worked on weekdays. it is a liststore widget where the user can edit the hours by clicking on the cell. The second widget represents exceptions when the user is not available to work like holidays or planned vacation. An entry can be added or removed by the +/- buttons at the bottom. There is a help button in the corner again. The help page would explain that the calendar is used for the estimation of completion date of the project.

Only the user can update her own calendar. The default values are the standard working calendar: 8 hours Monday - Friday, 0 hours on weekends, no exceptions. The default values

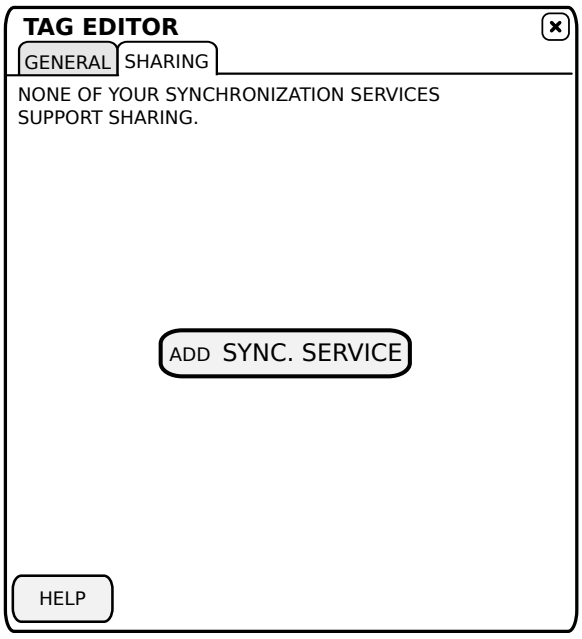


Figure 3.2: If there is no synchronization service with the support of sharing, the user is encouraged to add a new one.

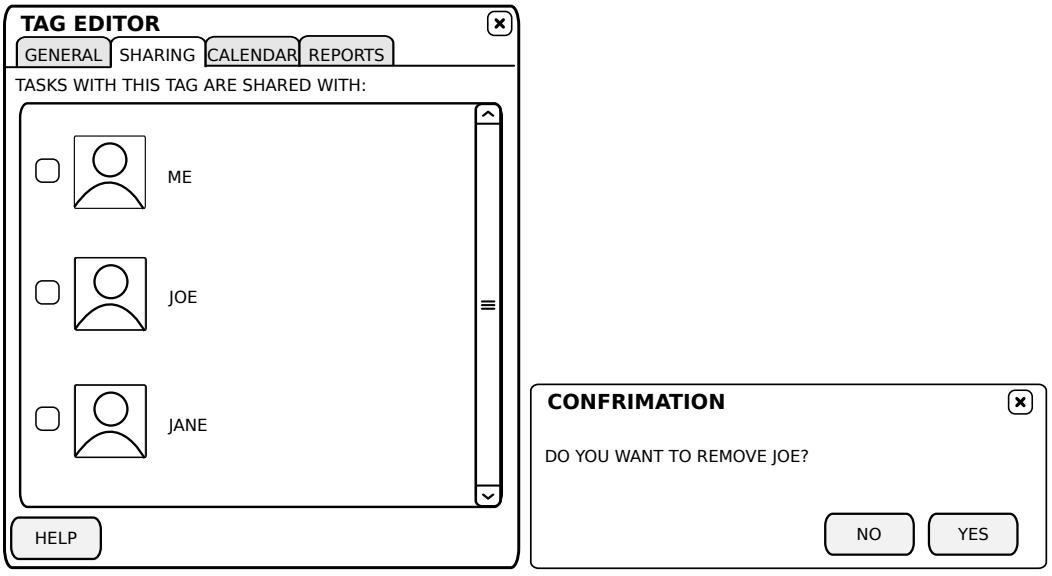


Figure 3.3: The content of the tab *Sharing* when a synchronization service with sharing supports is activated (left). Confirmation dialog to remove a contact „Joe“ (right).

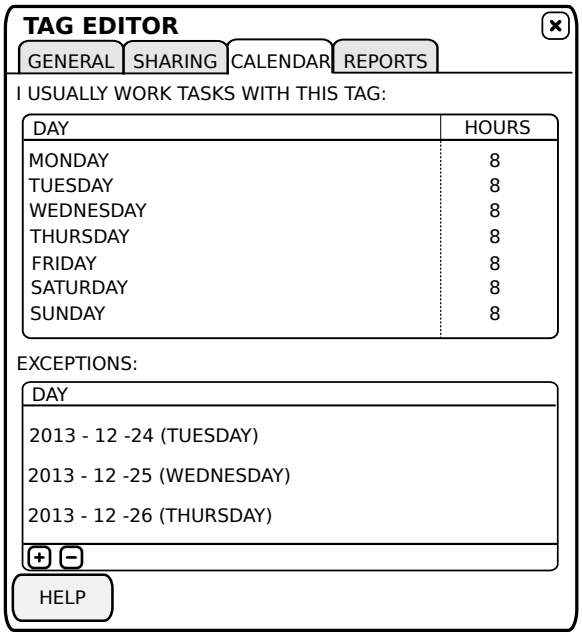


Figure 3.4: Tab *Calendar* can set the amount of hours working per workday. The user can list the exceptions like holidays or vacation.

are appropriate for fulltime jobs.

Tab *Reports* (Figure 3.5) allow team members to see reports on the activity of the team as whole. The user can specify the time range of the reports: last 7 days, this month or all time. The report consists of the time spent working during the period, work estimated to complete the project, time left to complete the project and the completion date. There are data about the progress made in the period aggregated to:

- *reports on tasks* – for every tasks which somebody worked on during the period, there is the current status of the task, the list of persons who worked on the task and the amount of time they spent working on it, and
- *reports on members* – for every team member, there is a list of tasks she worked on during the period with the amount of time spent and the status of task.

A task editor gets a new combobox where the user can assign the task to a member of the team. If she wants to remove the assignment, she can assign the task to a special value nobody. When the user opens an editor of a task without estimation entered by the user, a special bar (Firugre 3.6) with the prompt for the user estimation until the user enters it and close the editor. GTG nags the user for her estimation until she enters it. The estimation is represented in a simple language used in the alternative tools like Microsoft Project: a number followed by the unit. For example 1 day is `1d`, 4 hours is `4h`.

If the user is assigned to the task and the editor of the task is opened, the time tracker is activated. The progress of the tracking is displayed in a special bar (Figure 3.7) with the overview of time spent today, time spent in total, estimated time for this task.

The button *Report* in a editor opens a dialog with the report for the task. There could be found the overview of the estimations for the task done by all members of the team

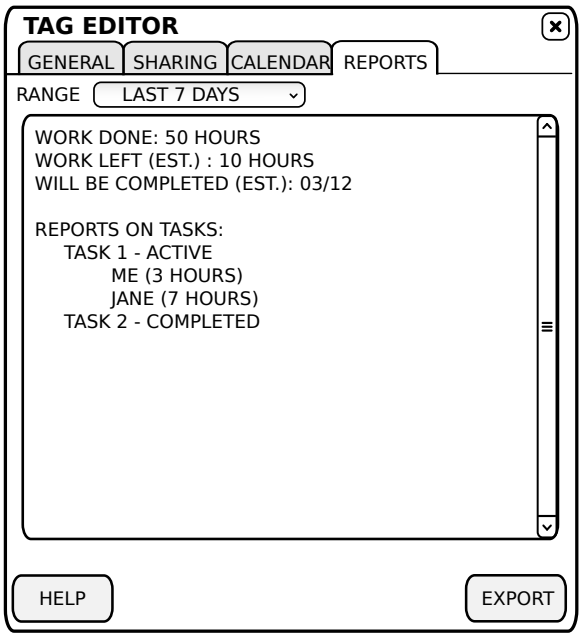


Figure 3.5: Tab *Reports* provide the overview of the activity given the time range.

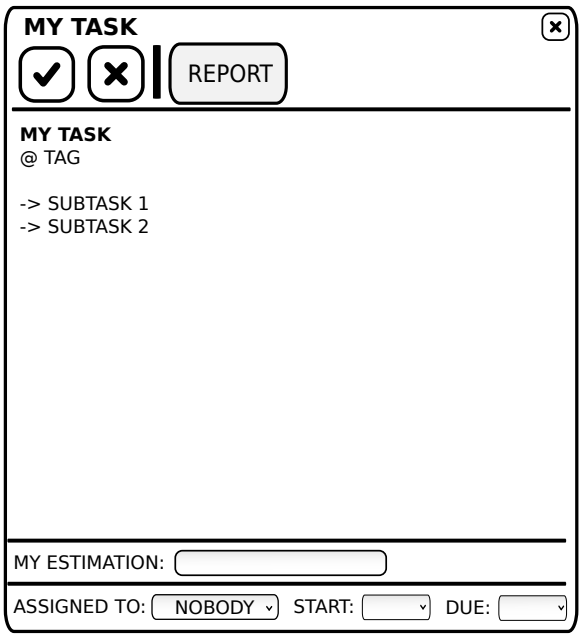


Figure 3.6: A task without an estimation nags the user to enter it.

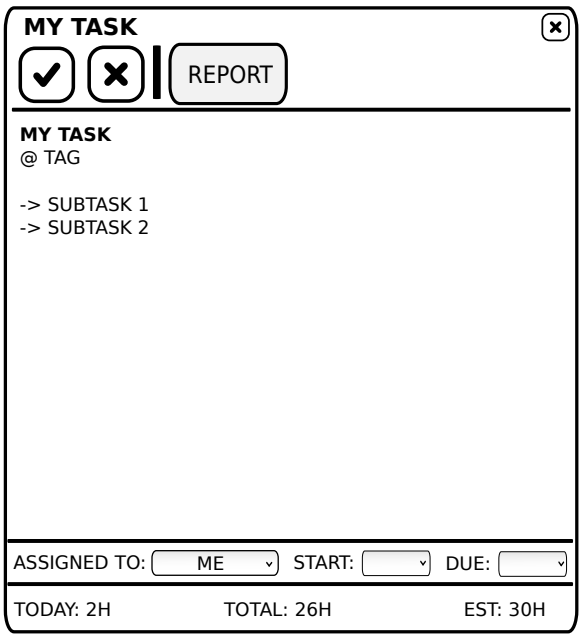


Figure 3.7: A special bar indicates time tracking.

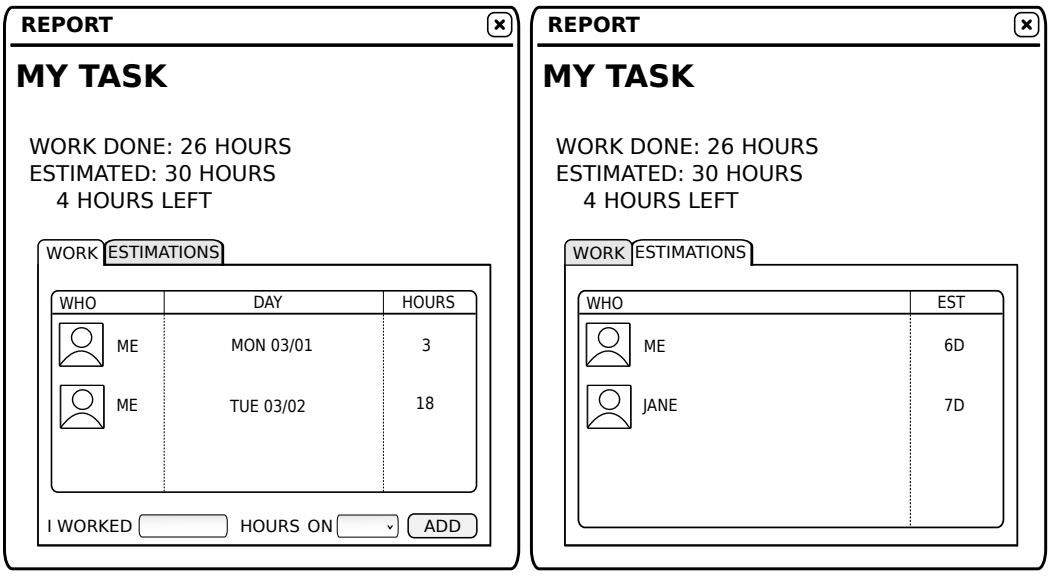


Figure 3.8: Report on the task. Work entries which could be edit on the left, the overview of estimations on the right.

(Figure 3.8 left). There are also entries about the time already spent on the task. (Figure 3.8 right) Every entry contains the member who worked on it, day and amount of the time spent. Presenting the member who worked on the task is useful if multiple members of a team work on the task over the period of the task. If the user is assigned to the task, she can update the entries or add a new one.

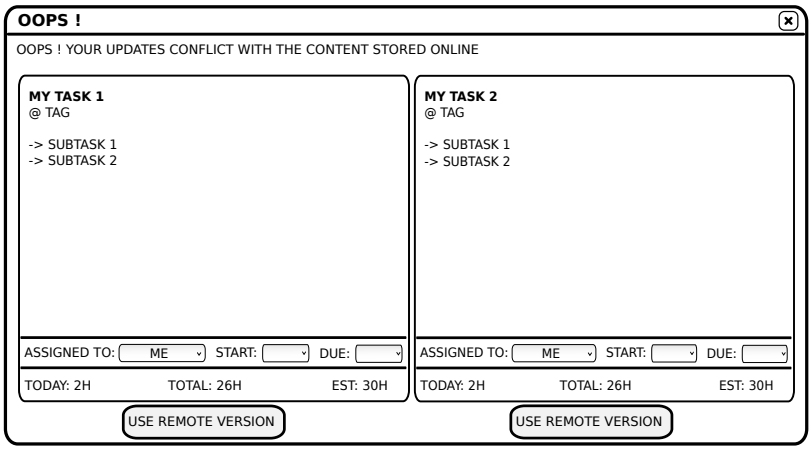


Figure 3.9: Dialog for solving conflicts when the user was offline

GTG can notice if the user is without the internet connection by querying Network Manager[1] . When the user goes offline, the synchronization is disabled until she is back online. If there are conflicts between changes done offline and changes done by other member, a dialog for solving conflicts is shown. (Figure 3.9) The dialog displays remote version of the task and the local version of the task which can be edited to merge changes. The user can decided which version is correct by clicking on the button below a task version.

[1]the manager of the network connections used in desktop environments like GNOME and KDE

Chapter 4

Implementation of the System

This chapter describes the implementation of the design proposed in Chapter 3. The chapter consists of describing tools used for the implementation followed by the description of extended infrastructure needed for the system. The synchronization of tasks via an extension of XMPP protocol and extension of the user interface are described in detail as well.

4.1 Tools Used for Implementation

As the purpose of this work was to extend GTG, the existing tools were used to implement the system. The extension was coded in Python language. PyGTK library was used for creating the user interface. My work added to the list of dependencies only a library for handling XMPP communication. The library is optional; if it is not installed, GTG fails gracefully without providing additional features.

GTG code guidelines require additions to the project to comply with PEP8 coding style. Uniformly formatted code, conventions for naming variables, and spacing style makes it easier to comprehend the code both for contributors and maintainers. The compliance with the style is checked with the tool `pep8`. My implementation complies with the coding style and `pep8` yields no error or warning.

Another requirement for new code in order to be accepted is passing through a simple code checker `pyflakes`. Pyflakes produces warnings for unintended side effects as importing a module without using it or initializing a variable without using it. Pyflakes yields no warning for my implementation.

GTG uses `GNU Gettext` from Python standard library for the internationalization of the user interface. My implementation follows the effort and all strings visible to the user are translatable.

4.2 Extension of GTG Infrastructure

My goal when extending GTG was to be independent of the underlying technology. I did not wanted the implemented system for cooperative project planning to be dependent of XMPP synchronization. It is easy to imagine synchronization based on another technology like own synchronization protocol. Therefore I put more effort into the extensibility of the solution.

GTG already went in this direction with its synchronization services. Each of them has a defined interface to communicate with. However, it was build only for synchronizing tasks and has no concept of people standing behind the tasks.

4.2.1 Concept of Contacts

A synchronization service can provide a list of people you can share tasks with. The list can be obtained via method `get_contacts()` of the object representing the service. A contact is defined as a triple `(person_id, name, avatar)`.

The first element `person_id` identifies a person uniquely. There are almost no restriction on the format of `person_id` but it should be a string. A service can decide what to use as the identifier. For example, it can be a numerical ID or JID[1] as used for XMPP synchronization. `person_id` is expected to not contain character `;` which is used for serialization of a list of `person_id`.

The second element `name` contains a name of the contact. It should be in human readable as it is displayed in the user interface for easier identification.

The last element `avatar` represents an user avatar. Together with the name it creates a visual identity of a user. Not all services might be able to provide user avatars or some of the users may not have a set avatar. Thus the element is only optional. The element is a string containing a file path to the image or in case of not set avatar python value `None`. This approach leads developers of services to cache images on the hard drive. Avatar can be requested couple of times in a short time span and downloading it on demand would be slower than loading it from the hard drive.

If a service supports the concept of contacts, it is required to return a list of contacts of the length at least one. The required contact is a special contact representing the user herself. It would not be satisfactory to return only available contacts for sharing with it. For example when assigning a task, the user might want to assign it to herself.

The special contact representing the user is often treated differently in the user interface. To find out which of those available contacts represents the user, a service is required to provide `get_user_person_id()` method. It is expected that user's `person_id` is represented as one of the contacts available via the service. Another aggregation method is `is_user_person_id()`. It is a predicate which tests if `person_id` represents the user.

If a service does not support the concept of the contacts, both `get_contacts()` and `get_user_person_id()` returns value `None`. It was implemented as the default behavior for all services. Thanks to inheritance from the generic service, all services support this extension of the interface.

There could be more synchronization services enabled at the time and the aggregation of contacts is required. It has been done by expanding object `DataStore` which provides interface between services and the rest of GTG. Methods `get_sharable_contacts()` and `get_user_person_ids()` aggregate contacts from all enabled services which support the concept.

4.2.2 Consequences of Contacts Aggregation

Aggregation of contacts has interesting consequences. `DataStore` does not retain the origin service of a contact and it is transparent to the rest of GTG. With compatible services it would be theoretically possible to share the same cooperative project via multiple multiple

[1] Jabber ID

instances of service or multiple services based on a different underlying technology. In other words, it could be possible to have a project on XMPP server and a web based project management tool. The implementation of multiple services for such a deployment it is outside the scope of this work and is let for future extension of GTG.

The extensibility have a negative consequences. The changes in the user interface have to consider multiple services and multiple identities for the user. An example of such a issue is a selection of person_id of the user. The user can have enabled multiple services and have multiple identities. Any person_id from get_user_person_id() could be used but a service has to *correct* identity to the user's one. It has to merge identities.

4.2.3 Extending Tag Object

As proposed in Chapter 3, a cooperative project is represented via a tag. Because of it, I extended the interface of Tag object for methods to set and get the list of contacts working on the project. `set_people_shared_with()` sets the list of people who work on the project and `get_people_shared_with()` retrieve the list. In both cases, the list consists only of `person_id`. The consequence of this is that when displaying a list of people who can be assigned to a task, it is required to obtain all sharable contacts with their details and filter people who can take part on this project. There is also a predicate `is_shared()` which indicates if the tag represents a cooperative project. It comes handy when deciding if an element of the user interface should be hidden or not.

`set_buddy_sharing()` is a convenience method which sets if the user represented by her person_id should or should not be a part of the team. The programmer does not have to care about retrieving the list of participant, changing a single value and passing it back.

4.2.4 Extending Task Object

The interface of Task Object had to be extended as well. It got couple of new fields. The simplest one is the field for `assignee` which represents person_id of the team member assigned to the task. Besides a standard getter and setter, there is a convenience method `is_assigned_to_me()` which changing the way user interface works. Another handy method is `is_shared()` which aggregates all tags assigned to the task and checks if at least one of them is shared.

As proposed in Chapter 3, a task can track the time spent on the task and keep estimations of the duration from people working on the project. A record about time tracking contains:

1. person_id of the team member who worked on the project,
2. day of the work, and
3. amount of time spent on the task expressed in number of whole minutes.

A record of estimation has only two fields: person and her estimation of the duration of the project. Because of the added new data, the XML representation of Task object had to be changed by adding new elements. The parser for Task object is very benevolent and ignores unknown elements. Thanks to that, it is possible to improve keep backward compatibility; all of the new elements are optional and older version of parser would ignore them.

4.3 XMPP Synchronization

For the implementation of the synchronization of tasks was selected XMPP[2] for the reasons described in Chapter 3. Main reason is that the implementation of the synchronization does not have to deal with the issues as authenticating users, encryption of the communication or many others aspects of communication itself. The details are not important for the purpose of this work.

The synchronization uses the already existing extension `XEP`[3] `0060: Publish Subcribe` [16] also known as PubSub. The extension allows to have object oriented pattern designed *Observer* over XMPP protocol. Participants in the communication can create their nodes and other participants can subscribe to the nodes. When there is something that participants in the protocol would like to announce to everybody else, they publish the content to the node. XMPP host server immediately announce the change to all subscribed participants about the new content. A node consists of several published items that can be retrieved if requested.

For example, when Romeo the user[4] would like to announce to the all interested people in the world, that he loves Juliet, he would create his own node and everybody interested can subscribe to the node. When Romeo decides to announce the big message, he easily sends a request to the server. Server, which could be represented by old gossip women, would broadcast the message to everybody interested; everybody who usually talks to the women. If somebody is interested in an older Romeo's message, the women can deliver it.

An example of practical usage of PubSub is announcing new syndicated content via PubSubHubbub[5]. The server of content provider owns a node and publish the new content on the behalf of the user. News feed readers are immediately notified about the new content and can pop up a notification for the user. The positive consequence of PubSubHubbub is the removing the need for the poling. The readers do not have any need to poll the server for the new content because they are immediately informed about new messages.

4.3.1 PubSub for XMPP Synchronization in GTG

When synchronizing tasks in GTG, the architecture of PubSub can be taken advantage of. A synchronization node represents a project. A project consists of several tasks which are represented as items. The architecture is depicted in Figure 4.1. A GTG client creates a PubSub node which represents a cooperative project. All GTG clients that are part of the project including the owner of the PubSub node subscribe to the node. It ensures that any change is propagated to every connected client.

When the user makes a change in a GTG client, the client publishes the updated task to the PubSub node of the project on the behalf of the user (Step 1). XMPP server accepts the change of the item, the GTG task, and announce the change to every connected GTG client (Step 2). The synchronization works on basis of Server-Client model where a client has full copy of working data. It allows the user to view and manage tasks offline—one of the proposed requirements in Chapter 3. When a library for handling XMPP and PubSub communication is used, GTG has to only implement reactions to the events.

[2] Extensible Messaging and Presence Protocol
[3] XMPP Extended Protocol
[4] The policy of XEP proposals is to use Shakespeare references for examples of the communication.
[5] https://code.google.com/p/pubsubhubbub/

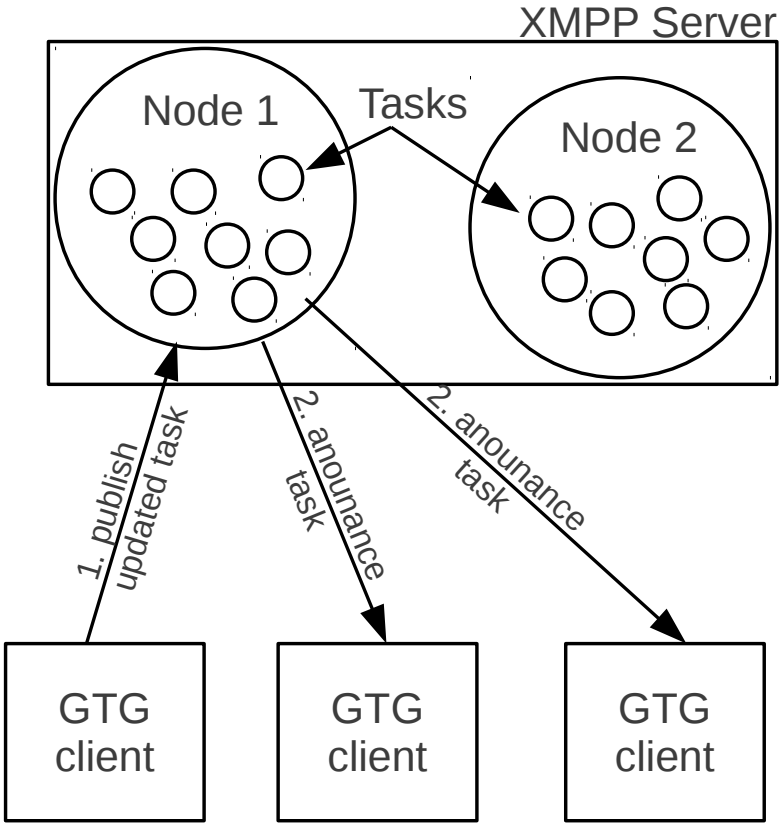


Figure 4.1: PubSub architecture adapted for needs of XMPP Synchronization in GTG

4.3.2 Tools Chosen for Implementation

There is a variety of available XMPP libraries for Python which are listed at the website of XMPP Standards Foundation[6]. After counseling Mr. Peter Saint-Andre, a co-founder of XMPP Standards Foundation and the author of [17], I decided to go with SleekXMPP library that is featured in [17].

SleekXMPP [8] is an MIT licensed XMPP library for Python. SleekXMPP is designed to be easily extended with as many XEPs as possible in the form of plugins. It comes with the support for PubSub out of the box. Another useful feature is parsing a XML message into a dictionary object. The parsing works similarly as ORM[7] that is explicitly defined for each plugin. The feature has a steep learning curve and the mapping is not intuitive. Fortunately, the programmer can introspect the parameters of the objects using Python tools.

There are also several existing implementations of XMPP Server[8]. After counseling Mr. Lionel Dricot, the external consultant of this work, I decided to use ejabberd server for the referential implementation. According to ejabberd's webpage [1] it is „distributed fault-tolerant Jabber/XMPP server in Erlang". Ejabberd is one of popular implementations.

Although my implementation was tested with Ejabbered, the intention was to lay down

[6] http://xmpp.org/xmpp-software/libraries/
[7] Object-relational mapping
[8] http://xmpp.org/xmpp-software/servers/

as few requirements for XMPP server as possible. The final implementation works with standardly packaged Ejabbered properly configured as described in Appendix A. No non-standard plugins or modification of Ejabberd code is required.

4.3.3 Overview of Implementation of XMPP Synchronization

XMPP synchronization is implemented as a standard synchronization service for GTG with the infrastructure extensions described in section 4.2. The code of the service is placed at `GTG/backends/backend_pubsub.py` containing two objects:

- `Backend` implementing the interface of a synchronization service,
- `PubsubClient` encapsulating details of SleekXMPP library.

The synchronization is based on the communication between those two objects representing the meeting point between GTG and XMPP side. Real-time collaboration of users is out of the scope of the synchronization service. It means that it is not possible multiple users edit the same task in the same time. The collaboration was not recognized as a crucial feature needed for cooperative project planning in Chapter 3. It would require massive changes to GTG infrastructure and rework of the task editor from scratch.

The intention while implementing the synchronization was the smoothest user experience possible. It means working on following problems:

- solving synchronization conflicts,
- preventing self-changed loop,
- storying content of tasks in persistent items,
- maintaining permission model,
- auto-discovery of relevant nodes, and
- synchronization of non-tasks data.

4.3.4 Solving Synchronization Conflicts

When dealing with the synchronization, I tackled the problem of synchronization conflicts. Such a conflict happens when two users are updating the same part of the project. In order to keep the user interface as possible, the conflicts are solved in the last write wins fashion. If multiple people edit the same task, the last update is valid. However, in the moment when one user publish the change, the other user receives a notification about the change. If the other user has been the changing the same task, the changes are preserved and later publish to the XMPP server.

When the user is offline and manages her tasks, GTG can't publish the changes XMPP server because of no internet connection. During the time spent offline, other users might have made big changes to the project. Changes might be very important and overriding some of them is not admissible. Therefore, when the user is back online, instead of propagating changes made offline, the user is asked to resolve conflict manually via the dialog for solving conflicts depicted in Figure 3.9.

To correctly handle offline support, `Backend` object persistently keeps its state from the finite state machine in Figure 4.2. A newly added synchronization service starts in

Start state. The service connects to XMPP server based on its configuration. If it succeed, provided configuration is valid and proceed to online mode. If not, the service raises an error to the user waiting for new configuration. *Online* state is the regular state when changes are published and changes of others are received. In the moment when network connection goes down or XMPP server is not available, the service changes its state into *Offline*; changes are not published, changes of others can't be received. When the network connection is up and XMPP server is available, the service enter *Solving conflicts* state. The service downloads all current tasks from the server and compare them with the local tasks. If version from the server is different from the local version, the dialog for solving conflicts is shown. If the connection is down again, the service is back in offline mode again. When all conflicts are resolved, the state changes back to *Online*.

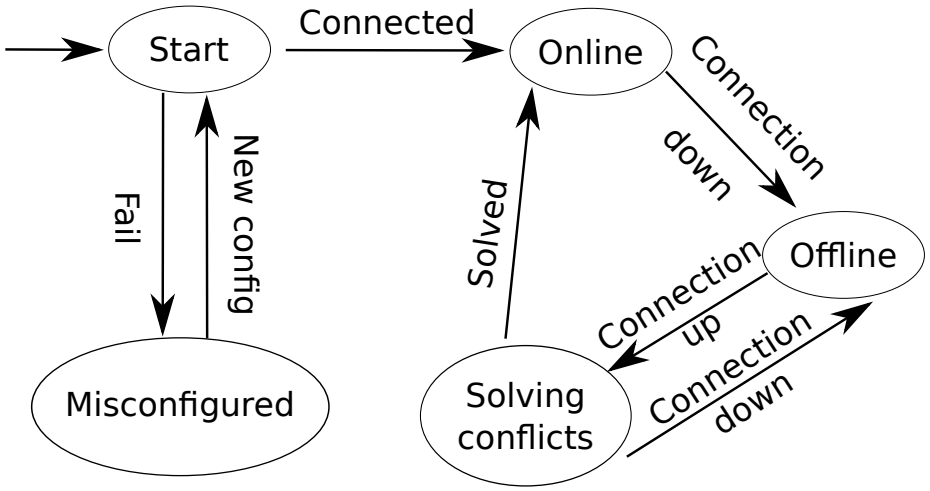


Figure 4.2: Finite state machine that facilitates the implementation of offline mode support

4.3.5 Preventing Self-Changed Loop

Every member of the team, every GTG client wants to be informed about changes that other clients publish. All GTG clients are therefore subscribed to nodes containing GTG tasks. When one of them publishes a modified task, every subscriber including the publisher is notified about modified task. Subscribers pass the changed task to GTG that in the similar way notifies every submodule of GTG including the synchronization service and creating a loop. Such a situation is in Figure 4.3:

1. Backend object has a new version of a task and passes it to PubsubClient.
2. PubsubClient publishes the task to appropriate Pubsub node.
3. The changed task is propagated to all subscribers including PubsubClient.
4. PubsubClient passes the task to Backend as it expects it to be changed.
5. Backend stores the task in GTG.
6. The task was modified as it comes from outside and is propagated to all submodules including Backend.

7. Backend passes a *modified* task to PubsubClient and next iteration starts.

The issue of the loop is not solved in PubSub as it is not a standard use case. It is expected that client can't be publisher and subscriber at the same time. My first attempt to solve this problem was was to store timestamp of the last actual modification of the task. However, I found out that the modification timestamp is also changed when the task is opened in the editor and closed afterwards; timestamp changes quite frequently. Ignoring changes to the task for couple of seconds since the last timestamp was problem because of trashing regular changes.

Another solution which I used in the end is to have a set of modified tasks. When a task is received (Step 4), Backend checks whether the task is in the set of modified tasks. If yes, it removes the task from the set but discards the task; it breaks the loop. If not, it put it into the set and later prevents Step 2 to happen.

In other words, after passing a changed task through the interface between PubsubClient and Backend, it is expected to get a notification that is throw away. The technique prevents the loop.

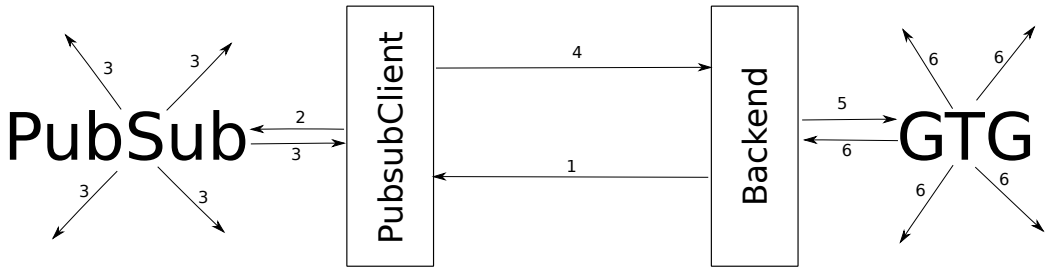


Figure 4.3: Sequence of passing changes that leads into a loop

4.3.6 Storying Content of Tasks in Persistent Items

Purpose of items in nodes in PubSub standard is enable clients to *catch up* with the current state of the node. If a client is not online for some time, it still can manually download all the newest content. A PubSub implementation can allow storing of those items persistently, not only for a short amount of time. Persistent items can be stored as long as they are needed. They can be even updated or deleted.

In accordance with the standard use case where PubSub node is only a mean of synchronization, the count of items that could be stored in one node is limited. Standard configuration of Ejabberd allows only 10 items per node. When no space is available for a new item, the oldest item is removed to make space for the new one. The oldest item is the item which has the smallest modification timestamp. In other words, if an item is modified, its modification timestamp is set to a current timestamp at the time and becomes the newest item.

One reason is that usually clients are not interested in past items but only in few of the newest ones. Another reason is the prevention against storing huge files or many small files inside PubSub. Administrators are safe against attacks where the attacker overloads server's database with useless data just to make the server run out of the available disk space.

On the other side, GTG has an edge use case where all items are needed, not only the newest ones. Without them server does not have the full copy of all tasks that are shared. There are three cases when the full copy on the server is needed:

1. New team member added to the project does not have any of shared tasks; full copy is needed.
2. GTG clients logs to the server after starting GTG. Client needs to fetch all changes that occurred while GTG has been turned off. Only couple of changes is not enough.
3. GTG clients returns from offline mode. It needs to obtain all tasks and discover conflicts between changes made offline and changes of other users.

With the insufficient low number of persistent items per node, GTG can easily have inconsistent data against other. With the fact in mind that the intention of the implementation is to limit changes to the server to bare minimum, administrator of the server is asked for specific configuration. She can raise the limit of maximal items per node in the configuration of XMPP server. If the limit is set high enough, node can store each item needed.

As future work, the server can be modified to change the maximal count of persistent items based on business logic. In case of a commercial hosting of space for tasks, the maximal count of persistent items could depend on the price plan. Another proposal might be to automatically detect GTG nodes and raise their limit from the side of server. For now, a suitable configuration from Appendix A is expected.

4.3.7 Maintaining Permission Model

As proposed in Chapter 3, the system uses a simple permission model. Every member of the project can create, read, update and delete tasks and also can add new members or remove existing ones. Nobody else can do anything with the project.

There are several available access models in PubSub:

- `Open` – default model; open to any changes and subscribing,
- `Presence` – used mainly for instant messaging systems,
- `Roster` – allow access to a specific group or groups from user's contact list only,
- `Authorize` – the node owner must approve access, and/or
- `Whitelist` – access is limited to a list of users.

Authorize can't be used because the new member had to ask for the membership herself and afterwards she must be authorized. It would yield too much bureaucracy and the beauty of the simple permission model would be lost. Roster is close to the expected behavior. A contact has already had all teammates in one or more groups. However, it would fail on the situation where the user adds a new user and she adds her colleague which is new to the team. The original user does not have the new colleague in her roster and therefore she can't be added to the project until it gets into the roster. In other words, everybody had to be in rosters of owners prior adding to the team.

The final implementation uses Whitelist access model. The list of users is manager by assigning them affiliations to the nodes. As far as every member has all permissions like the owner of the node, all members have to be owners of the node. PubSub standard does not support notifying users that they are added to or removed from a whitelist. It is more difficult to implement the missing notifications, but it can be done by polling. Every couple of seconds request all affiliated noted to the user. If there is a new node in the list, subscribe to it. If a node is missing, stop synchronizing it.

4.3.8 Auto-Discovery of Relevant Nodes

There are two possible ways how a member can be added to a team: either the member applies to be the member of the team or the other way around, only a member who is already part of the team can add new teams. As proposed in Chapter 3, the implemented system should use the latter option. It reduces steps of adding a new member from 2 (requesting, authorizing) to only 1 (just add the member). Also the potential member does not have access to the existing project; she can't even find out that there is any project.

The advantages come with the necessity to auto-discover all relevant nodes to the user on startup. Algorithm 1 shows the pseudo code of auto-discovery. The algorithm uses the fact that GTG nodes have whitelist access mode and every member has to be owner of a node. There must be an affiliation between the user and the node. Pubsub standard provides a useful request which can return the list of affiliated nodes of the user. For each node, the configuration of node is checked. It is important to set the maximal count of persistent items to as high number as possible. The access mode has to be whitelist.

It is important for the user to update list of all team members which can be obtained as list of affiliated users to the node. If the user is not yet subscribed, subscribe her. This situation can happen on the first discover of the node. It is worth to mention that PubSub has support for multiple subscription for each node which is not desired behavior.

All GTG nodes follow the naming convention where the name starts with the prefix `GTG_` and the rest of the name is UUIDv4[9]. An example of such a name is `GTG_68414580-856d-4118-9bfd-c268fb9ca508`. The name convention helps administrators to easily identify GTG nodes and prevents multiple instances of GTG from collisions when choosing a name of node. It is not possible to identify a node only by the tag it represents as there might be multiple teams with the same name. Information about the tag the node represents is stored in the configuration, parameter title in form `Project @tag` for tag `@tag`.

4.3.9 Synchronization of Non-Tasks Data

Besides the tasks which belong to the project, there are other data that could be synchronized. For example, parameters of the tag which the project is associated with. The parameters allow the user to choose the background color of the tag and its icon. If it is synchronized, every team member would have the same color associated with the tag. I have made decision to not share the parameters of the tag. Reason is that some members might want to use different colors and icons to match the rest of their tags' colors and icons. Every user can set the color of tasks as she wants and likes.

Another kind of data that could be synchronized is calendar: How much time does the user regularly spent on the project on given day? When does she plan to take holidays?

[9] Universally unique identifier, version 4

Algorithm 1: Auto-discover all available nodes and ensure the proper setup

Input: connection (*a connection to the PubSub server*)
Output: nodes (*a list of of nodes and their parameters*)

```
nodes := find all affiliated nodes using connection ;
for each node in nodes do
    configuration := request configuration of node using connection (node);
    if configuration is not set properly then
        set proper configuration using connection ;
    end
    node.tag := find project tag (configuration) ;
    node.members := find team members using connection (node) ;
    if user not subscribed to the node then
        subscribe to the node using connection (node) ;
    end
end
return nodes;
```

The calendar is used for the estimation of the project finish date. It is shared as a special item which id is `calendar`. There can't be any synchronization conflicts on this item because of member is the owner of its own calendar. Every time the calendar change, data for other users are changed but not for the current user. It's calendar is the one that counts.

4.4 Implementation of the Extension of the User Interface

The user interface was extended as it was proposed in Chapter 3 without any major differences from the proposed wireframes. After consideration of how much of the new user interface should be add I abandoned the intention to create the changes as a plugin for GTG. However, I tried to hide as many new elements as possible so the regular user who does not want to use the cooperative project planning, has almost the same interface as without the extension. In this section I would like to highlight some of the implementation details of the user interface.

4.4.1 Tag Editor

The code for Tag Editor was originally placed as `GTG/gtk/browser/tag_editor.py`. With the implemented extension the editor got lot of new options and the code much longer. I moved the editor out of into its own submodule into `GTG/gtk/tag_editor`. Each new tab in the editor is represented by it's own object, for example settings for Calendar are represented by the object `TagCalendarPage` in file `GTG/gtk/tag_editor/calendar_page.py`.

In the proposed wireframe of sharing tab 3.2 there is a button that is used to add a new synchronization dialog. There was method only to open the settings dialog for synchronization service but no shortcut to directly add a new synchronization service. Thus I implemented a new method for directly adding a new service by opening the dialog and automatically clicking on *add* button. It works now as it was intended.

Dealing with Python's day-of-week Problem When implementing Calendar tab 3.4 I tackled a problem. There is no method in Python standard library which returns the first day of the week based on the local settings of the computer. The expected values for a user in USA is that first day Sunday, for a user in Czech Republic is Monday.

I submitted a feature request to Python issue tracker[10]. When I was contemplating whether to contribute a patch for the issue or not, I came to conclusion that it would not have the direct effect on this work. Creating a patch, make it through code reviews into Python official repository and releasing it would need a long period of time to be able to use it to solve the problem in GTG. I managed to find a workaround in the codebase of project hamster which is published under GPL licence. The workaround calls unix command `locale` and from its output computes the first day of the week based on locale. In the case that `locale` is not available, it falls back to Monday as the first day of the week.

The report tab provides reports based on the given time range the user want. When generating a report, GTG iterates over all tasks with the tag it generates report for. For each task it notices how much time was spent on that task and who worked on that task during that time. In the end, data is aggregated into two views:

- time spent by the task, and
- time spent by the member the team.

The estimated date of the end of the project is computed by critical path method (CPM). The method taskes the directed acyclic graph of tasks which are associated with the tag and find the duration of the critical path. For its purposes it uses estimations for each task. If a task does not have an assigned person, for the matter of calculation the task could be done by anybody who is available at that time. The calculation is connected with the calendar of all members in the team.

4.4.2 Task Editor

Task editor got new features which are hidden unless task is shared. The user can assign some member to the task and also give her estimation. I decided to not go with the three point estimation in the first version of the system but just one point estimation. The estimation is accepted as a simple expression like `5d 4h` which is converted to minutes expected to spend on the task. For the conversion the following relations are used:

- 1 day = 8 hours,
- 1 hour = 60 minutes

User's estimation is multiplied by her personal coefficient which represents how does she underestimate or overestimate on average for the average person who works on the task. Coefficient 1.0 means perfect estimation, 3.0 means that she underestimates by the factor 3, 0.2 means that she overestimates by the factor 5. The coerced estimations are taken into account: the final estimation is their average.

To see the progress on the task, the time spent on the task is tracked. The tracking happens only when the person who is assigned to the task has opened task editor. The tracking is implemented in a different way than a usual time tracker like project Hamster; it is enough

[10] http://bugs.python.org/issue17659

for them to just mark start and the end of tracking. As GTG is not interested in when the time happens only by how much time the assignee spends on the task. GTG has to save the status of time multiple times to prevent lost of data if the program crashes.

When the assignee starts tracking the time, the start time is saved into a temporary variable. If it is the case that the user has already worked on the task before, the start time is moved back in the time by that amount. In other words, the time is tracked as if the user works on the task without break. Every minute is trigged a timer and the difference between the current time and time of start in minutes is saved as work for today. The same code is ran when the task editor is closed.

Task reports dialog which is placed in file `GTG/gtk/editor/report.py` provides the overview of time spent on the task and also estimations of members plus global estimation for the task. If the user has worked on a task without GTG, she can manipulate the tracked time.

4.5 Merge Request

The implemented system was proposed to be merged into trunk, the official repository of the project from where it will be released in the next version of GTG. The merge request is at the address
https://code.launchpad.net/~izidor/gtg/collaborative-gtg/+merge/165007.

Chapter 5

Usability testing

In order to validate the proposed extension of GTG and its implementation, I organised a usability testing that is described in this chapter. The tested use case, the form of the testing, and my observations are described. In the end, the identified issues and their solutions are discussed.

5.1 Form of the Usability Testing

The purpose of the usability testing was to find out how real users use the application, whether they are able to accomplish common tasks, and identify unintuitive steps in the process. The testing was done in person to facilitate observation of the user and her instant reactions to the user interface. There were 17 participants, all of them students in the age group of 20-25. They are part of the targeted user group as most of them will probably have or already have an office job where the cooperative project planning is a must. Many of team have experienced the non-trivial team work while working on school projects. They represent the targeted users as described in Chapter 3.

One testing session took about 15-25 minutes. A participant used a preconfigured computer. Almost none of the participants had an prior experience with GTG. Because the extension is aimed at experienced users of GTG, the participant was shown a small demonstration of the standard features first. The basic concepts were introduced including a showoff some existing plugins and synchronization services. Afterwards the purpose of this work, enabling the experience of GTG for individuals to small teams with cooperative projects, was explained. The participant was given the use case situation proceeded through a series of expected steps for the cooperative project planning. The steps went through all new user interface added to standard GTG. In the end, the participant expressed her feelings and gave the general feedback.

5.2 Tested Use Case

A participant was asked to imagine the following situation. You have seen amazing things that a 3D printer can print. With a couple of friends you decided to build your own 3D printer as a hobby, weekend project. As the project is quite challenging, you decided to use GTG to plan and keep the track of the project. Please follow these steps:

1. Enter couple of tasks that need to be done to build a 3D printer. All of tasks should have tag `@printer`.

2. Add a new synchronization service based on PubSub which is extension of XMPP. Your credentials are `tester@jab.izi` as the username and `password` as the password.
3. Share tasks with the tag `@printer` to couple of your contacts.
4. Assign tasks to your team members.
5. Enter the estimation for each task.
6. For the estimation purposes, you want to inform GTG that you will work on the project only on weekends.
7. A friend of yours working on the project with you was irresponsible. She has no time for the project couple months and has not done anything at all. Remove her from the project and assign her tasks to other members of the project.
8. Look at the time reports of one of your tasks. You find out that you forgot to track it. Add a record about working 20 hours in past.
9. Find out the estimated delivery date of the project. Export the report about the project to a file.

5.3 Observations

Step 1 It was pretty easy to complete after the initial demonstration. All of the participants created one task with several subtasks. About the half of the participants were confused about the lack of any confirmation button that would explicitly save the task. The auto-save feature of GTG was not mention while the demonstration. When talking about it in the feedback part, the affected participants told that they are not used to automatic saving but they could get use to it.

Step 2 Although the dialog for adding a new synchronization service was presented in the initial demo, the participants took some time to open it. They were thinking for some time before deciding to open the dialog. I think that a possible explanation might be that it was early in the testing session and they were getting used to the interface. A participant was confused about the wording of the button OK (Figure 5.1 left). She was afraid to click on it because it would accept it without configuration which is not the case. According to her, `Add` would be a better caption. The second step was harder. Some of the participants wanted to use *Tags to sync* widget for sharing `@printer` tag. A few of them were confused about the position of *Enable syncing* button that was expected to be in the bottom of the dialog. The dialog is the standard part of GTG infrastructure and its user interface was not changed for the extension.

Step 3 The participants took some time to open the tag editor for sharing settings. Work with the sharing tab was without problems. As the participants mentioned in the feedback, adding a menu item to the context menu for tag would make it easier to find the sharing settings.

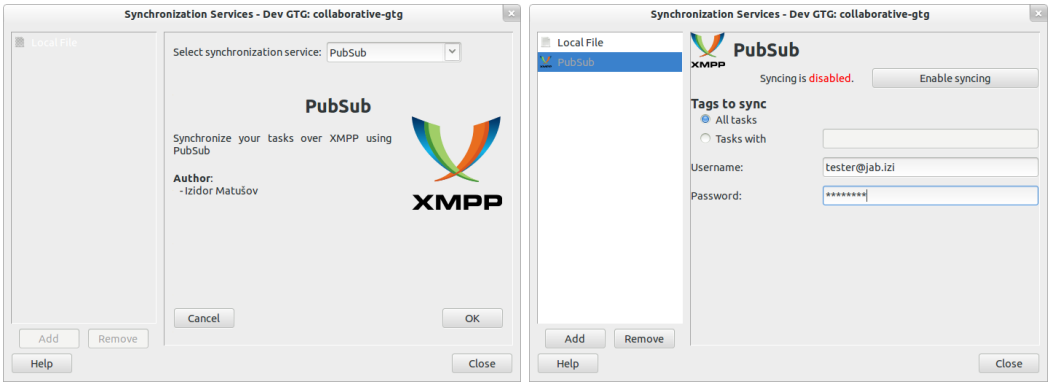


Figure 5.1: Two standard screens used to add a new service and its configuration.

Step 4 The participants were looking for the option from the context menu of task in the main window of GTG. The option for assigning people from the context menu was not implemented. Some of the participants were looking for some control elements in toolbar and menubar. Eventually, they found out the assignee menu directly in the task editor.

Step 5 Most of the participants set start date and due date of tasks. It was obvious they were trying to fill the form elements because they considered them as required. The participants were unsure what format was the expected for the estimation field. When encouraged to find it out on their own, the red font color for a non-valid entry was helpful. All of them came with the expected format `<amount> <unit>`. However, most of them used full words for units instead of short variant, it means `4 days` instead of `4d`.

Step 6 Finding the calendar widget was quite hard for the users. According to the proposed wireframe (Figure 3.1 right), all projects settings including the calendar are available via `Edit tag` menu item. The participants discarded this option because they did not want to edit the tag but change the calendar. In the feedback, they said that a direct item to calendar would make them to find it easily. The editing of assigned hours in the calendar was cumbersome because editing numbers required exactly a single click to start editing the number.

Step 7 Removing a person from the project was an easy step so down in the process. For the second part of the step, participants were trying to sort tasks by the assigned person. In the feedback, they asked for integration with the search feature where they just enter the name of the participant to show all tasks a person is assigned to. Another proposed way how to satisfy their need was to have a special column for assignees besides start date and due date. Thanks to the information next to the title of the task, they easily oriented which task was assigned to which person. However, it is not possible to sort by assignee in that way.

Step 8 The participants were again looking for a menu item in the context menu where they could not found it. The button for task reports is placed in the toolbar of the task editor. It was a new place where an additional element was placed. As the users were getting used to the user interface, they were looking for old places. Although they scanned

the toolbar in the main window, they did not scan the toolbar in the task editor. Adding a new entry was without a problem.

Step 9 Although the participants were told to about imaging subtasks as rather dependencies instead a task being a sum of all subtasks, the participants were expecting to find the general report when opening their main task and pressing report. An additional factor might be that the previous step included the same window and the participants expected tasks to be connected. Again, *Edit tag* menu item was not convincing them to open it to find general reports. The rest of the step, finding out the delivery date and exporting it to the disk was easy for them.

5.4 Discussion of Identified Issues

The biggest usability issue was *Edit tag* menu item from the context menu of a tag. The label did not convince the participants that there might be sharing, calendar settings and project reports. After adding additional menu items for each tab, it was much easier to discover features. The items did nothing more than opened the tag editor dialog and switched to the tab directly.

It was surprising to see how participants were opening the context menu (right-click menu) each time that the control element was not displayed in the user interface of the main window. Some of the participants even used the context menu when opening the task editor instead of double-clicking on the task title.

Another big issue was the calendar which was hard to use because of the clicking policy for GtkTreeWidget. After getting used to it, the participants were able to complete the step, but it was unintuitive for them. The widget was later reworked to improve its usability. Other minor issues like sorting and searching by assignee, settings assignee via context menu, and opening task reports from context menu are left for the future work.

The participants as new users of GTG found potential usability weakness of standard GTG. The biggest issue was automatic saving of tasks. The participants were not used and were afraid to close a task without confirmation. One of GTG intentions of GTG is the implicit saving. Another issue was the standard dialog for synchronization services which was not change for the purpose of this work.

At the end of testing, the participants got used to the system of the user interface elements. They were efficiently working with GTG and eventually accomplished all required steps. In the feedback, they appreciated the simplicity of the user interface and even admitted they would get use to the auto-save feature.

Chapter 6

Conclusion

In this work, the extension to Getting Things GNOME (GTG) was proposed, designed, implemented and tested.

The motivation why it is woth to plan and manage a cooperative project via a task manager like GTG was discussed. The existing solutions for cooperative projects were explored and took into the consideration for the extension.

The target audience for the extension are companies where people work in small teams and communicate via XMPP. The companies have the access to its own IT infrastructure. Simplicity, transparency and trust are important features of the companies' culture. Those values are mirrored in the requirements of the extension.

The needs of the target audience were analyzed and solutions to satisfy their needs were proposed. The users need to share and manage their tasks, assign tasks to co-workers, manage teams, estimate the future of the project and work offline while on business trips. This work does not aim at solving the sharing files associated with tasks.

Wireframes of the proposed extension were presented with the commentary of intended user experience when working with the extension.

The proposed extension was implemented as three submodules: changes to GTG infrastructure, as a new synchronization service and extended user interface. The intention was to change as less standard GTG code and its user interface as possible.

With the changes to GTG infrastructure a concept of contacts were introduce. If a synchronization service implements the concept and remote server supports it, cooperative project planning can be synchronized over multiple technologies not only XMPP & XEP 0060: Pubslish Subscribe. Model of Task was extended for the concept of assigned person who works on that task, time tracking and estimation tracking. Model of Tag was extended on the concept of team which works on the project.

The new synchronization service uses the changes to the GTG infrastructure to support synchronization of tasks among members of the same team. The service is based on XMPP and its extension XEP 0060: Publish Subscribe. The user management, encryption of the communication, and exchange of messages is delegated to XMPP. Basic principles of XEP 0060: Publish Subscribe were explained and how they affect implementation decisions. As a client library for XMPP was used SleekXMPP that is the only added and optional dependency. As the referential implementation of XMPP server and its extensions was used Ejabberd.

The user interface was implemented almost identically to proposed wireframes. Almost all new elements in the user interface are hidden if the user does not use sharing and cooperative project planning. Visual changes for the regular user who uses GTG only as

a task manager for individual. The usability testing revealed couple of usability issues with the user interface, mostly connected with discovering features. The issues with the biggest impact on the usability were already fixed. The implementation was proposed to merge into the official repository of GTG. From the repository it will be released in the next version of GTG.

As the future work, another synchronization service which would support the concept of contacts could be implemented. There might be added some special business logic to server code base that would remove the limitation of the maximal count of persistent items per node. Multiple techniques for estimations could be implemented and the team can choose a technique they want to use. There are left some additional features which might improve the usability of the user interface like allowing sorting tasks by assignee.

Bibliography

[1] ejabberd — distributed fault-tolerant Jabber/XMPP server in Erlang [online]. `http://www.ejabberd.im`, 2011 [cit. 2013-05-19].

[2] Homepage of Getting Things GNOME! [online]. `http://gtg.fritalk.com/`, [cit. 2013-01-02].

[3] 37Signals. Homepage of BaseCamp [online]. `http://basecamp.com`, [cit. 2013-01-02].

[4] David Allen. *Getting Things Done: The Art of Stress-Free Productivity.* Penguin Books, 2002. ISBN 01-420-0028-0.

[5] John Bartlett and et al. *Project Risk Analysis and Management Guide.* Association for Project Management, 2004. ISBN 19-034-9412-5.

[6] Paulo Cabido. GTG 2009 Summer of Code's wiki page [online]. `https://live.gnome.org/gtg/soc2009`, 2009 [cit. 2013-01-02].

[7] Lionel Dricot. Getting Things Gnome! 0.1 – 'Just 5 minutes more' [online]. `http://ploum.net/post/206-getting-things-gnome-01-just-5-minutes-more`, 2009 [cit. 2013-01-02].

[8] Stout Lance Fritz nathan. SleekXMPP: Homepage [online]. `http://www.sleekxmpp.com`, 2011 [cit. 2013-05-19].

[9] Gregory T. Haugan. *Project Planning and Scheduling (Project Management Essential Library.).* Management Concepts, 2001. ISBN 15-672-6136-1.

[10] Luca Invernizzi. Getting things GNOME integration with online services [online]. `https://live.gnome.org/gtg/soc2010_invernizzi`, 2010 [cit. 2013-01-02].

[11] P. Frederick Jr. Brooks. *The mythical man-month (anniversary ed.).* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995. ISBN 02-018-3595-9.

[12] Jean-Philippe Lange. Homepage of Redmine [online]. `http://www.redmine.org`, [cit. 2013-01-02].

[13] Izidor Matušov. Story of Getting Things GNOME [online]. `https://www.youtube.com/watch?v=9k4Rl33eaaA`, 2012 [cit. 2013-01-02].

[14] Microsoft. Homepage of Microsoft Project [online]. `http://www.microsoft.com/project/`, [cit. 2013-01-02].

[15] Microsoft. Homepage of SharePoint [online]. `http://sharepoint.microsoft.com`, [cit. 2013-01-02].

[16] Peter Millard and et al. *XEP-0060: Publish-Subscribe.* Version 1.13, XMPP Standars Foundation, 2010-07-12.

[17] Peter Saint-Andre, Kevin Smith, and Remko Tronon. *XMPP: The Definitive Guide Building Real-Time Applications with Jabber Technologies.* O'Reilly Media, Inc., 2009.

[18] Chris Schlaeger. Homepage of The TaskJuggler Project Management Software [online]. `http://taskjuggler.org`, [cit. 2013-01-02].

[19] Fog Creek Software. Homepage of Trello [online]. `http://trello.com`, [cit. 2013-01-02].

[20] Joel Spolsky. Evidence based scheduling [online]. `http://www.joelonsoftware.com/items/2007/10/26.html`, 2007 [cit. 2013-01-02].

Appendix A

Installation and Configuring XMPP Server

As described in Chapter 4.3.1, Ejabberd was chosen as the referential XMPP server implementation.

Ejabberd can be easily installed on the server machine from repositories of most distributions:

```
# Debian/Ubuntu based distributions
sudo apt-get install ejabberd
# Fedora
sudo yum install ejabberd
# Archlinux
sudo pacman -S ejabberd
```

Source code can be downloaded from http://www.ejabberd.im.

The intention of this work is to work with minimal configuration. The only change against the default configuration is the higher maximal count of items per node. It can be done using the following command:

```
sudo sed -i 's/{mod_pubsub,.*/&\n\t\t{max_items_node, 9999},/' \
    /etc/ejabberd/ejabberd.cfg
```

Make sure your configuration have:

- enabled whitelist access mode,
- enabled `mod_pubsub`,
- the limitation `max_items_node` is high enough,
- desired `<domain>` is enabled; localhost is enabled by default.

A new user can be added via the following command

```
sudo ejabberdctl register <name> <domain> <password>
```

Adding contacts can be done in a regular XMPP client like Pidgin or Empathy.

Be sure that `<domain>` and `pubsub.<domain>` addresses are resolved to your server machine. For example, if you are using localhost, add following lines to your `/ect/hosts`:

```
localhost 127.0.0.1
pubsub.localhost 127.0.0.1
```

For running GTG just from the sources on DVD you need following libraries if not already installed:

- SleekXMPP available from https://github.com/fritzy/SleekXMPP,
- Liblarch available from https://github.com/liblarch/liblarch,
- PyGTK,
- python-configobj,
- python-xdg,
- python-dbus.