Bizzilla: A Collaborative Task Management Environment with Expert Finding

Jeremy Debattista, Charlie Abela
Faculty of Information and Communication Technology,
Department of Intelligent Computer Systems,
University of Malta,
Msida MSA 2080, Malta
{jdeb0015,charlie.abela}@um.edu.mt

Abstract. Researchers put enormous amounts of time and effort into collaborative tasks, such as writing up research proposals or papers. Typically, for such tasks, collaborators use an ad hoc combination of different tools, from version control software, latex editors, instant messaging and emails, to keep track of the tasks status. Time is inevitably wasted in pooling together the various resources and to keep them synchronised, as well as to monitor the status of the subtasks assigned to the different collaborators. Furthermore, it is at times required to quickly seek expertise advice from colleagues about particular aspects of the ongoing research task. In this paper we present Bizzilla, a collaborative task management environment with expert finding capabilities. Bizzilla is based on a plug-in architecture and provides a suite of tools that allow for collaborative task management, real-time collaborative editing, automatic resource synchronisation, instant-messaging and expert-finding based on the idea of collaborative opportunism.

Keywords: Real-time Collaboration, Task Management, Expert Finder, Collaborative Opportunism

1 Introduction

One tool researchers normally use for collaboration and sharing of resources are Versioning Control Systems (VCS) which allow collaborators to download the latest copy of the needed resource on their PC. The only way collaborators can interact together using the VCS is by emails or instant messaging (IM) which will cause inefficiency when searching for previous to-do lists or any other relevant information mentioned in the past. Also from time to time, a to-do list has to either be manually updated or else downloaded from the VCS if available as a resource there.

Tools such as Versioning Control Systems are not adequate for collaborative tasks. The main reason is that if two collaborators require to access and edit the same shared resource, the system only allows one to have the lock and thus edit. This is not feasible in a collaborative environment. A possible solution for this
problem is the inclusion of a real-time collaborative editing systems (RTCES),
similar to those in [6, 3] which apply various techniques discussed in [6, 12, 16, 14,
15, 13]. Another problem is the merging of files, where conflicts could arise when
merging edited resources. Synchronization is also a problem with such tools since
shared resources have to be manually updated before use in order to make sure
that one is working on the latest version of the resource. In this paper, we will
look on the idea of having a collaborative environment where a researcher is able
to create collaborative tasks.

Sometimes researchers would need to find some persons with the relevant
expertise in the area being studied. Various experts can be found for the differ-
ent research areas, but sometimes it is not possible to connect with some expert
if there are no social connections. Therefore it would be desirable to find ex-
erts which one has some relationship with, such as colleagues, students, and
other collaborators who have worked with the researcher in question on previous
work. This is known as collaborative opportunism. Zhang et. al. in [19] already
addressed this issue and their approach gave promising results. In this paper, we
also focus on how collaborative opportunism can be achieved through the use of
FOAF\(^1\) profiles and link analysis on the relationship between candidate experts,
ranking candidate experts using the HITS algorithm as used in [8, 4, 1, 18].

The rest of the paper is organized as follows. In section 2, we discuss the
design issues for our system, starting off with a top level description and archi-
tecture of the prototype. The prototype implementation is described in section 3.
In section 4, we will discuss the results of our prototype and finally we conclude
the paper with some remarks in section 5.

## 2 Design

The aim of this project is to have a system which allows a user to create col-
laborative tasks and to allow the system to find possible collaborators according
to the task. The system will be modelled on a client-server architecture and is
depicted in fig. 1.

The backbone of our prototype will be NEPOMUK\(^2\). With NEPOMUK,
resources and files are semantically annotated and information for every file is
logged [11]. In this project we are mainly concerned about the PIMO and TMO
models.

For the environment being investigated, the idea of client-server models can
be adapted. Thus with such a setup, the PIMO, tasks and their resources are
in a centralized storage space which is synchronized with the client’s desktop
through some communication between both sides.

The Collaborative Editing Module will consist of an application which im-
plements various interfaces which will eventually serve as a collaborative editing
tool for Bizzilla. This module will allow various users using the same application
to connect with each other and collaborate on the same collaborative resource.

\(^1\) [http://www.foaf-project.org/](http://www.foaf-project.org/)

\(^2\) [http://nepomuk.semanticdesktop.org](http://nepomuk.semanticdesktop.org)
A server will act as a negotiator of the shared session; handling all notifications and operations between clients. We will also develop a collaborative \LaTeX editor plug-in for this environment.

Xpert Finder will make use of information retrieval techniques and the HITS algorithm. This module will be split into two main stages. The first stage is the retrieving and downloading of publications from the FOAF profile repository. These will then be indexed. The second stage is the retrieval of candidate experts.

In this section we first discuss the main design issues for the collaborative task environment, then we move on to the Xpert Finder design issues.

### 2.1 Bizzilla - Collaborative Task Environment

We decided that this application will have NEPOMUK as an underlying framework since it will manage automatically the user’s PIMO whilst with Bizzilla we update the TMO. The Bizzilla environment will allow a set of collaborative plug-in tools and the main interface with the user will be a sidebar. These collaborative plug-ins will be connected to a web service provided by Bizzilla in order to allow collaborative real-time editing. We are adopting the idea of having plugin applications which are accessible from Bizzilla since eventually various users of this system will be able to add and use their own collaborative tool and are not restricted to only one tool.

Bizzilla will be connected to the NEPOMUK SSD, ensuring that all tasks are updated in the user’s PIMO. When creating a new task, Bizzilla will inform the
task creator about any on-going task in the range of the start date chosen to start
the collaborative task, according to tasks noted on Bizzilla and other tasks noted
on other applications using the NEPOMUK framework. Bizzilla is also connected
to a broker application on a server which will handle all task synchronization
between collaborators. This broker make also sure that the clients are informed
by any pending task invites.

The sharable component interface will include the properties and classes
which are needed to create a sharable component in line with our backend OT
Engine. The interfaces described are part of the collaborative editing module in
which other classes related to the Concurrency Control function will be defined.
The interface for this component should be abstract since we need to cater for
various tools and not just one type. This module will be connected to a web
service which will handle all the shared session operations, such as the session
management, discovery of client and notifications.

Bizzilla should be an application which is always running on the user’s desk-
top, since it will manage all the collaborative tasks created in the environment.
Due to this property, Bizzilla should not interfere while the user is working on
other tasks. The UI design of Bizzilla should be in a form of a sidebar with an
auto hide property. The sidebar will handle all information about the tasks a user
is currently participating in. It will provide detailed information on every task
and, similar to Kasimir [7], the sidebar changes its view according to what task
is currently being accessed by the user. The environment will have an in-built
to-do list, instant messenger and auto-update features for the tasks.

2.2 Xpert Finder

Xpert Finder will rely on a repository of FOAF files and publications. This
module will be split into two stages. The first stage is the pre-processing stage
where the FOAF profiles in the repository are parsed to retrieve all publications,
downloaded and indexed. For this prototype, we will be doing this stage only
once. The second stage is the expert finding stage where experts are ranked
according to a user’s query. In Bizzilla, the idea of finding experts is based on
collaborative opportunism.

The main aim of this module is to find a number of experts related to a
particular area. To achieve this aim, we will make use of a repository of FOAF
profiles from which publications will be retrieved and downloaded from the in-
ternet. We decided to use FOAF files in order to find candidate experts in a
research area. This will be called the generic Xpert Finder, which will consider
experts from all the FOAF profiles in the repository. This generic framework will
then be expanded in order to create a collaborative opportunism model from the
FOAF profiles. This idea enables the personalization of expert models according
to the FOAF file of the user in question. For this module we will separate the
pre-processing stage and the expert finder stage.

\[3\] downloaded from http://www.arnetminer.org
In the pre-processing stage, we will parse the FOAF files using the Jena\(^4\) framework. We are mainly interested in the `<foaf:Publication>` class at this stage. At this stage we need to store the publications together with the author’s name prior to indexing and searching. With all FOAF profiles parsed, we would then need to download any publications which are not available on our repository. The final step of this stage is the pre-processing stage, where we will build an inverted index out of the downloaded publications using information retrieval (IR) techniques such as vector space model. We will also make use of a stemmer to build our index in-order to improve the IR performance in the second stage.

A two step approach similar to [19] is adopted in the expert finder stage. At each step, candidate experts will be given a score and finally they will be ranked in descending order. We will refer to the first step as the Candidate Selection and Initial Score. The second step will be called the Candidate Authority Score, where we will be implementing the HITS algorithm to calculate the final score similarly to [1, 8]. A slight modification in the Candidate Selection and Initial Score step will be needed in this step in order to satisfy the personalization needs. The HITS algorithm is ideal for our implementation since it allows us to analyse the link structure of the expert’s graph. This is important within our collaborative opportunism Xpert Finder model: edge directions are relevant to determine the type of relationship between vertices. According to [1], HITS gives high precision values for an expert search but low recall values for such a search. However, this is not a cause for concern since overridden by the high recall value from the initial IR search.

3 Implementation

Bizzilla is based on the .NET framework due to the beWeeVee\(^5\) library and the Windows API which allowed us to create a sidebar with the desired autohide property. For the collaborative LaTeX environment we created a windows form application which includes various functionality with regard to LaTeX keywords, syntax highlighting and compiling. As for the editor component, we made use of a Windows Presentation Framework control, since a more flexible environment than the traditional windows form controls is needed to create a collaborative workspace.

Bizzilla links a user with both NEPOMUK and the web server. Since it is connected to the NEPOMUK social semantic desktop, the data used in the application is semantically annotated, therefore meaningful. NEPOMUK offer a number of APIs to access its services however they are Java dependent. An XML/RPC\(^6\) protocol is used to connect and communicate with the local NEPOMUK server which stores all semantic information about the user including the PIMO and TMO. Bizzilla also connects to a broker application on the server.

\(^{4}\) http://jena.sourceforge.net

\(^{5}\) http://www.beweevee.com/

\(^{6}\) http://www.xmlrpc.com/
which manages task creation and synchronization, and expert finding in order to exploit the collaborative task management functionality.

3.1 Task Creation and Synchronization

Bizzilla provides functionality for task creation. The application allows a user to give as much detail to the task as possible. The task details are then used to create an instance in the TMO in NEPOMUK. When creating a task, NEPOMUK will take care of creating the URI and the task id for the task. The task id is used in the server to identify between the tasks. When a new task is created, we inform the broker about it. We make an exact copy of the new task details and resources on the server, which are then used for collaboration purposes. The broker handles the pending invitations by storing them in a hashtable. When the background thread running on the client requires to check for any existing pending invitations, the broker uses the hashtable which contains the required information.

Task synchronization is an important aspect in task collaborative environment. One needs to be informed by any new updates on the task such as involvement of new collaborators and updates of resources pertaining to the task. Each change made to the task is serialized to the task log. This task log is available both locally and remotely, with the remote task file being the most updated version. The approach we took ensures that for each task the local log file is checked against the remote log file and the differences between the two XML files identified. In this process, if the server file has more nodes than the local file, first we perform the necessary updates and then the local file is patched with the new nodes. We also check for any new collaborators which joined the task, since in that case, the client’s PIMO and TMO has to be updated with these new persons.

3.2 Real-Time Collaboration

For real-time collaboration we made use of BeWeeVee OT Engine\textsuperscript{7}. The project created by Corvalius allows easy integration of their library with other tools. We based our real-time collaborative approach on Jupiter [12]. The main function of the web service is to control notifications and clients in a shared session. We wanted to allow various concurrent real-time collaborative sessions. For this we had to keep track of all sessions, clients and notifications. We decided to store this data dynamically, using hash tables.

When a user opens a file, the client invokes a new session on the local computer. Here the Operational Transformation (OT) helper classes create the session and the necessary callback function on the client side. This function is used to open a communication line between the client and the web service which handles the notifications. Eventually the client connects to this service and signs in, passing information such as the user name. On sign up, the helper classes

\textsuperscript{7} http://www.beweève.com
request the web service if the session has already been started. A typical session is identified by \texttt{taskID\_fileName}. If the session exists, then the user joins a session. When a user joins a session, if that session is already open and editing has already been made, then all changes will be communicated to the newly joined peer, in order to keep her updated with the latest changes done in the current session. If the session is not yet created, then it is created on the web service. Session information will be returned to the client, which will be used in order to communicate changes from client to web service. When a user closes the session, the web service is informed and if there are no more peers for that session, the session is removed from the hash table.

During the life-cycle of the collaborative session, the web service receive a list of operations which need to be transformed on all collaborating sites in a particular session. When one or more operations are received by the web service, the service notifies all the clients in the session by passing the operation(s) through the client’s callback function. The operation(s) are then received on the client sites where OT is performed and each operation is executed on each of the connected clients. Each client tool has the beWeeVee OT algorithm implemented. The task of sending operations to client sites is done in parallel, thus everyone will receive the operations approximately at the same time. Clients are also notified of any failures.

### 3.3 Xpert Finder

For the expert finder we need a repository of FOAF files and publications. We downloaded a number of FOAF files from Arnetminer\(^8\). The variety of FOAF profiles were chosen carefully, that is, apart from trying to have a large search space of research areas such as Semantic Web, Real-Time Verification Systems and others, we also had in mind the evaluation of our system.

In the pre-processing stage we created a list of publications and their authors by extracting all publications from the FOAF files using the Jena infrastructure. In the extraction process, we would need to extract the data found in the \texttt{<foaf:name>} and \texttt{<foaf:publications>} properties. These publications and their authors are stored in a Hash Map which is serialized. The publication name is the key whilst a list of authors is the value. Having a list of publications, we needed to download them. Unfortunately, no free digital library\(^9\) grants access to their web services to search for publications through an external source, thus Google Search API was used as an alternative solution. The last part of this step is the indexing part. We made use of Lucene\(^10\) which represents documents as a vector space model. After creating the index, Lucene gives us the possibility to create an analyzer, either implementing stemming (we used Porter Stemmer) or not.

In the expert finder stage for the generic module, we will consider experts from all FOAF profiles in the repository. The \textit{Candidate Selection and Initial}

\(^8\) http://www.arnetminer.org

\(^9\) for example: http://citeseerx.ist.psu.edu/

\(^10\) http://lucene.apache.org/java/3_1_0/api/core/overview-summary.html
Score step requires some query which indicates the area within which experts need to be found. Using Lucene, we will query our inverted index to find publications similar to the query. The search yields a maximum of 1000 results. We decided to use a threshold on the top $K$ documents. $K$ is an arbitrary value which is used to define the desired number of top results returned from the query. After retrieving the documents according to a query, we will check the authors of these publications to gather a list of candidate experts. This is done using the hash map created in the pre-processing stage. For each candidate we assign an initial score which is the score given by the search result to the article in which the candidate in question contributed. This score is incremented by other scores if her contributions appear in other articles in the top-k publications. We will then return a list of candidate experts with their initial scores.

In the personalized module, we have did slight modification to the Candidate Selection and Initial Score step from the generic module in order to expand and create a collaborative opportunism model. This idea enables the personalization of expert models according to the FOAF file of the user in question. For the personalized module, in the step Candidate Selection and Initial Score, we first find candidate experts who have a direct or indirect (friend-of-a-friend) connection with the user in question, thus limiting the search space of possible experts. Here we build relations between the querying user and other possible experts using their FOAF profiles and \(<\text{foaf:knows}\>\) property. To find candidate experts further away from level one depth, we have created a recursive method which mines for experts using FOAF profiles and \(<\text{foaf:knows}\>\) property until the required depth level is met. By this, we are reducing our search space of possible experts to those who are directly or indirectly related to some user. All the experts retrieved will then have their publications extracted from their FOAF profile and stored in a temporary hash map. The rest follows the generic module, where when searching, only the candidates which are in the temporary table can be retrieved.

Having the list of expert candidates, we then use JUNG to create a graph of the relations between these candidates. This is the Candidate Authority Score step. We create candidate experts as vertices with directed edges (links) pointing between these vertices. For example if person A knows B, then we have a directed edge pointing from A to B. We find relations by using the candidate’s FOAF profile and the \(<\text{foaf:knows}\>\) property. The final graph will have its authority and hub scores calculated using JUNG’s HITS algorithm implementation. Having the authority and hub scores calculated, we then traverse each vertex and calculate the candidate’s final score. This is done by adding the authority value with the initial value calculated after the retrieval of publications.

4 Evaluation

We consider this stage as the most important of our research, hence we made sure to tackle all ambiguity well. We divided our evaluation into two parts; the collaborative environment evaluation and the Expert finder evaluation. For each
evaluation, we first look at some literature and similar systems’ evaluation. Then we proposed some experiments and evaluated the results.

4.1 Bizzilla Collaborative Task Environment Evaluation

In [10], a collaboration evaluation framework (CEF) was proposed in order to evaluate and assess tools in a collaborative environment more effectively. This framework provides a means to describe the characteristics of a collaborative process, metrics to evaluate tools in a collaborative process and guidance on changes to performance which can be made without effecting the cost. The authors build this framework on Thompson’s [17] three types of coordination and Clark’s [2] eight collaborative behaviours.

The authors in [9] proposed an improvement of Nielsen’s heuristics evaluation technique in order to take into consideration the collaboration aspect (as discussed in CEF) as well as the interface design. Collaborative software has to evaluate the impact of the tools in a collaborative process in order to have a complete evaluation of collaborative software. The methodology of such framework is similar to that of a traditional heuristic evaluation. The usability analyst (or any evaluator familiar with the tool being evaluated) will conduct the evaluation twice; once with the tool not in a collaborative environment and once with the tool in a collaborative environment. The collaborative software is evaluated on a set of heuristics, in which ratings, feedback and suggestions are given for each feature being evaluated.

The selection of participants is an important issue for evaluation. According to Dumas and Redish in [5], the participants should be interested in the software. In our case it would be suitable to find participants interested in collaborative task management and competent in \LaTeX\ skills. This will make the participants feel at ease during the task. They also defined that the number of participants should bound between 6 to 12 participants with 3 to 5 participants in each subgroup.

**Bizzilla Experimental Setup** We managed to find 4 persons who were split in pairs. In this subsection, we first look at the task given to all participants to do using the respective tools. We then give a brief overview of what tools have been used. Finally we compare the key findings. Each group was given a simple task. 

**Group 1** used Bizzilla Collaborative Task Management Environment which comes with an inbuilt Instant Messenger, \LaTeX\ collaborative editing plugin and task creation features. 

**Group 2** used Skype\footnote{http://www.skype.com} to discuss. LatexLab was used in order to do the report.

**Bizzilla Experimental Results** The evaluation, involved a task which each group had to created on their respective environment and a questionnaire related to the collaborative behaviours as discussed in [9]. Both environments used had
similar features, that is, an editor and a chat facility to discuss issues during joint tasks. The features in the environment used by Group 2 were catered for by distinct applications (LatexLab, Skype), as opposed to the Bizzilla environment which incorporates all required features in the sidebar. Both environments support the *mutual adjustment* coordination since they both support most of the collaborative behaviours. Overall, the Bizzilla environment gathers the basic required tools in a collaboration, thus it is more favourable than the other evaluated environment. Figure 2 shows the results from the evaluation. Some deficiencies in Bizzilla collaborative behaviours can be seen in transmission, notification and confirmation. This evaluation has helped us to point out these deficiencies in Bizzilla, which eventually would make the collaborative environment more complete and improved performance without effecting the costs, as pointed out by [10]. The mentioned deficiencies, however, can easily be improved by some changes to the environment.

![Fig. 2. Evaluation Rating Results](image)

### 4.2 Xpert Finder Evaluation

In the evaluation experiments done in the xpert finder, we conducted a number of experiments to evaluate the impact of the HITS algorithm on the score. We also evaluated the approximate best value for $K$ which is used in the Candidate Selection and Initial Score step, using precision and recall, as used in [1, 19].

**Xpert Finder Experiments** In our evaluation we run tests on both our Generic and Personalized Expert finder models. Our discussion is based on the phrase *Semantic Web*. The generic model was evaluated first. The first experiment was the evaluation of the precision and recall scores for different values of $k$ (the number of publications considered in the IR search to retrieve the candidate experts). We then perform a number of experiments on how different scores can affect affecting the candidate expert ranking. With these results, we
compare the different rankings found with the rankings provided in Saffron¹². The results given from various queries performed on the website we created are then compared with any feedback given (via a website created for evaluation). We then evaluated the personalized expert model by performing a search on the phrase Semantic Web on a FOAF profile. Different depth levels were considered for this experiment.

**Xpert Finder Results** In our first experiment, we found that our implementation value for $K$, that is 30, gives us the highest precision value but very low recall value. A high precision with low recall is not ideal. It is important to have a balance between precision and recall values in order to have a good amount of relevant candidate experts without leaving out any useful ones, whilst ensuring that there are few not relevant candidate experts. We conclude that a $K$ value between 50 and 100 would be more ideal in this situation, since we will get a bigger sphere of relevant candidate experts in the search area, but we will also get more candidate experts who are not relevant. Figure 3 show a Precision vs Recall graph for different $K$ values for query Semantic Web.

![Fig. 3. Precision vs Recall](image)

In our second experiment, we conducted tests where we showed candidate ranking according to the following scores: Initial Score, Initial Score + Authority Score, Authority Score. From this we concluded that the initial score + authority score gives us the best candidate expert ranking. The first score gave us the

¹² http://saffron.deri.ie/
ranking according to the initial search results from the publications, thus did not take care of the collaborative opportunism ideal. This ranked high any candidate experts which might not be experts on the subject or the expert’s publication ranked high. The second score gave weighting on the initial score, which boosted those experts whose authority score is high, and reduced the rank position of those with low authority score. The last score is based on the authority score. A social bias will play a big role here. Our expert finder managed to retrieve eight expert candidates from 18 possible candidates listed in Saffron. Figure 4 show different scores and ranking for each candidate for the query Semantic Web.

Fig. 4. Different Score Values and Ranking

The feedback given by the users from the website consisted of rating of the search results and ranking, their preferred ranking for the candidates being listed (for the query they performed), and also any suggested experts which were not listed. The users rank and suggest experts according their belief on who are the most suitable candidates in a particular area, mostly depending on personal work, publications read, and people whom they have heard about, for example in a conference. We had 128 hits with 32 unique visitors. A number of queries were irrelevant with respect to a research field such as names of persons and other terms such as “dog”. From these, the total number of correct searches

\[18\] is the total number of experts retrieved from Saffron and are present as FOAF profiles in our repository.
which added up to 103 queries, we got feedback from 19.47% of the hits, which amounted to 20 queries. From these hits, the users ranked the reliability of the Xpert Finder tool between average and reliable. In the last experiment, we found that the rankings given by the generic module with the personalized module for some user. The results were also good for different depth levels.

5 Conclusion

From the results of the evaluation, we came up with various indications as to where our prototype has led. We were quite happy with Bizzilla; from the evaluation it seemed that the participants found it quite useful for collaborative tasks even though we still need to improve on it. The results have shown that Bizzilla should improve in the aspect of sharing resources after a task has been created. Also, it lacks tools for task tracking. On the other hand, the grouping of various tools required for collaboration tasks such as the instant messenger and to-do list under one environment was a plus. As regards to Xpert Finder, we were also satisfied with the feedback given by the users who used our generic expert model on the website. The expert module is based on the idea of collaborative opportunism, thus results from feedback are based on the participants' idea of the ranked experts. Apart from the feedback we also calculated precision and recall values for the term Semantic Web. We managed to get a high precision and low recall with the K value of 30 for the k-top documents. This was not the ideal situation since a balance between both values would have given us a wider choice of experts without forgoing the precision value.

Overall, we are satisfied with what we have managed to achieve so far given the time and resource limitations. We have achieved acceptable results for five out of seven collaborative behaviors evaluated in the collaborative task environment. With regard to the expert finder module, the feedback was positive and we are confident that these results are encouraging for further research and development which would eventually be of use in the real world.

References


