HiveMind Collaborative Mind Map Editor: Architecture and Implementation of Network Subsystem

Andrey Vasilev, Oleg Kandaurov, Alexander Kulikov, Ilya Paramonov
Yaroslavl State University
Yaroslavl, Russia
{vamonster, kandaurovoleg, ivparamonov}@gmail.com

Abstract

HiveMind is a cross-platform mind map editor. Its most important feature is collaborative mind mapping, allowing different people to edit mind maps together regardless of their location.

In this paper we discuss the architecture of HiveMind network subsystem and implementation of collaborative mind map editing based on the XMPP pubsub extension protocol. Our approach operates in terms of abstract commands, which makes it loosely coupled from the application domain. It also makes possible to spread our teamwork architecture to a broader class of applications, which could benefit from collaborative document editing.

Index Terms: Mind map, collaboration, pubsub, XMPP.

I. Introduction

Mind map is a diagram used to represent words, tasks, ideas, or other items linked to and arranged around a central key word or idea. There is no predefined algorithm on how to add and structure these data, which makes mind maps suitable for such activities as study, project management, problem solving, brainstorming.

These kinds of work often involve many people. It reveals several issues. Firstly, all these people should be gathered in one place to begin collaboration. Secondly, it is hard to provide equal access to all participants, because only one person at the time can make changes to the mind map being constructed, for instance, on a whiteboard.

HiveMind is a cross-platform collaborative mind map editor, which main idea is to give everyone an equal opportunity to contribute to the shared mind map regardless of their location either by using a mobile device or a personal computer.

In general, the collaboration process can be described in the following way. One of participants publishes his/her mind map as a network service and goes on editing it. Another user connects to the service and retrieves the latest copy of the published mind map. After that, both users edit the map together. By changing a policy on the server side, it is possible to support for various teamwork scenarios.

To allow starting teamwork at any moment we needed to choose a message exchange system. We selected Extensible Messaging and Presence Protocol (XMPP), because it helped us to manage with several network-related issues [1].

In this paper we discuss the architecture of HiveMind network subsystem and implementation of collaborative mind map editing based on the XMPP pubsub extension protocol.

II. Network Subsystem Architecture

The XMPP protocol family is build out of two things: the core technology and the XMPP Extension protocols (XEP). The core is responsible for the message exchange between different parts of the system. Various extension protocols are built on top of the core. The process of creation and maintenance of such protocols is managed by XMPP Standards Foundation [2].

The XMPP publish-subscribe extension [3] uses the classic "publish-subscribe" or "observer" design pattern: a person or application publishes information, and an event notification (with or without payload) is broadcasted to all authorized subscribers. In general, the relationship between the publisher and the subscriber is mediated by a service that receives publication requests and broadcasts event notifications to subscribers.

Service may support several distinct entities available for subscription. These entities are called "nodes". There are different notification lists for each node hosted on the service. Publishers send data to the node and subscribers receive event notifications from it. Nodes can also maintain history of events and provide other services that supplement the pure pubsub model. Data send to and from the node is called "item".

In terms of publish-subscribe extension the collaboration process can be described as follows. A user creates a service and a single node for message exchange. The other participants subscribe to the created node. When anyone makes a change to the mind map it is sent to the service, which notifies all participants about the change. The data is sent in the form of the item payload. Having this principle scheme in mind, we began to form the teamwork protocol and implement it in detail.

Each change to the mind map is atomic. All changes to the mind map are done with the use of dialogs. When the edit operation is completed, QUndoCommand is created and pushed into the command stack. For every change command type XML serialization and deserialization functions are made. The use of edit commands causes the problem of sharing initial contents of the mind map. In order to execute identical commands, all participants must have exact copies of the initial mind map. In order to achieve this goal, the first item on the node must contain XML-serialized mind map. All the other items, as discussed earlier, hold serialized change commands. When a participant subscribes to the node he/she receives all the data stored in the node.

Each modification published in the node is recorded as a "changeset". The changeset holds information about type of change, time it was formed, and the author of the change (see fig. 1). Service may reject the changeset if it is irrelevant (e.g. user tries to update mind map node when it was already deleted by another user) and send error notification back to the subscriber.

Protocol design proposes asynchronous propagation of modifications on the client side of publish/subscribe interaction. Modifications to the mind map are transmitted to the service in the form of XML-serialized commands and are not stored in the local undo stack. The only changesets received via update notifications are added to the stack. So, the subscriber must wait for the correct notification from the service in order to see changes that he or she has made to the mind map.

The subscribers can retrieve all items stored in the node anytime. This capability is used to synchronize contents of their local copy of the mind map if transmitted changeset is rejected. Service does not need any additional effort to have up-to-date map because all notifications are local and cannot be lost during transmission.

To implement this asynchronous process, we introduced a new element to HiveMind core — NetworkController. It manages publish-subscribe protocol handlers. All QUndoCommands,

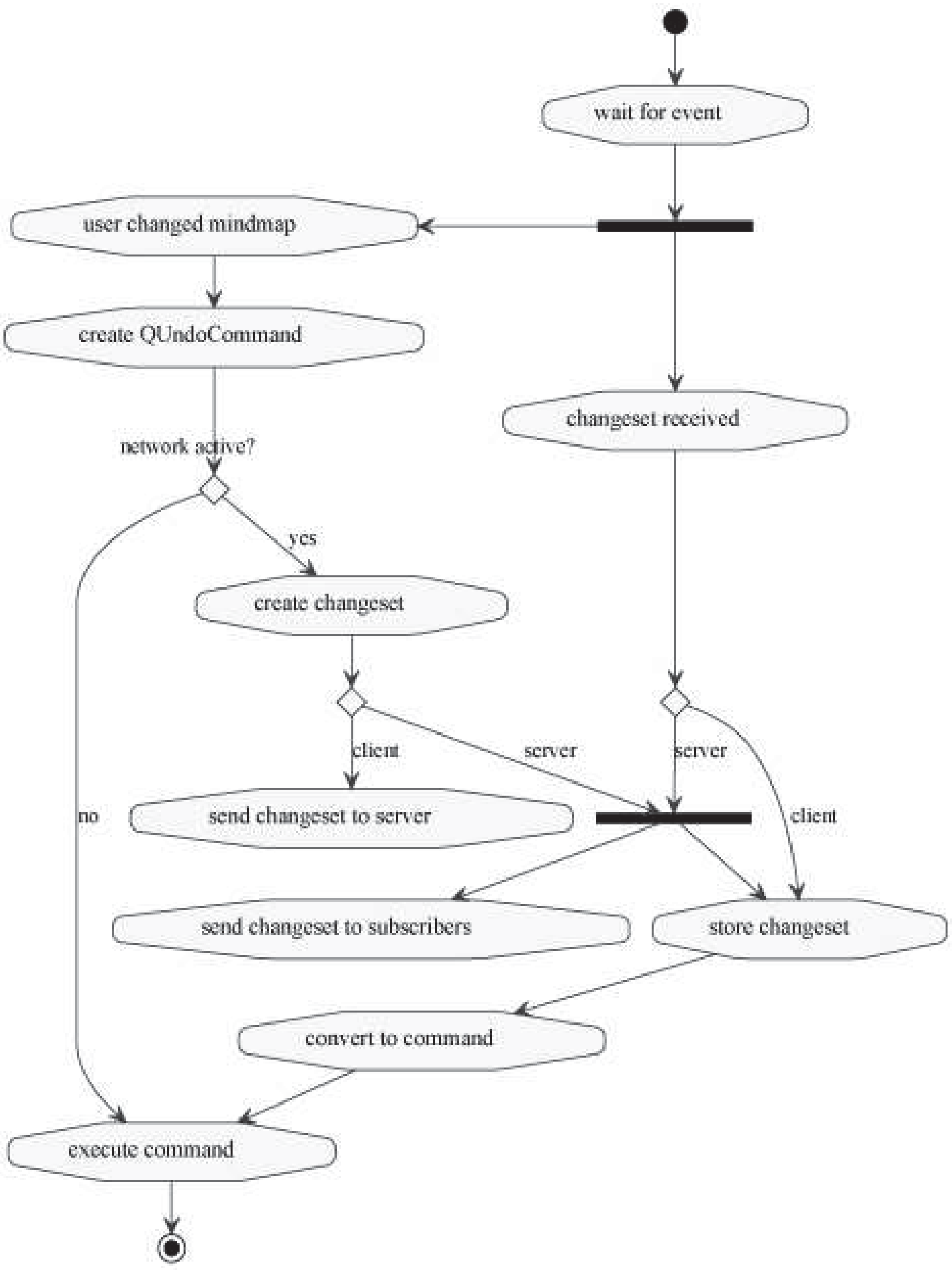


Fig. 1. Changeset propagation

created as a result of mind map modifications, are passed to the controller. If there is no network collaboration at the moment, the command will be added to the local undo/redo stack. Otherwise, command will be serialized and sent to the service. Update notifications are deserialized into QUndoCommands and added to the undo/redo stack.

III. Implementation of Network Subsystem

Network subsystem of HiveMind is implemented on top of Twisted [4] and Wokkel [5]. Twisted is a framework for writing asynchronous, event-driven network applications in Python. It has additional support for many GUI frameworks like Qt, GTK, Tk, and others.

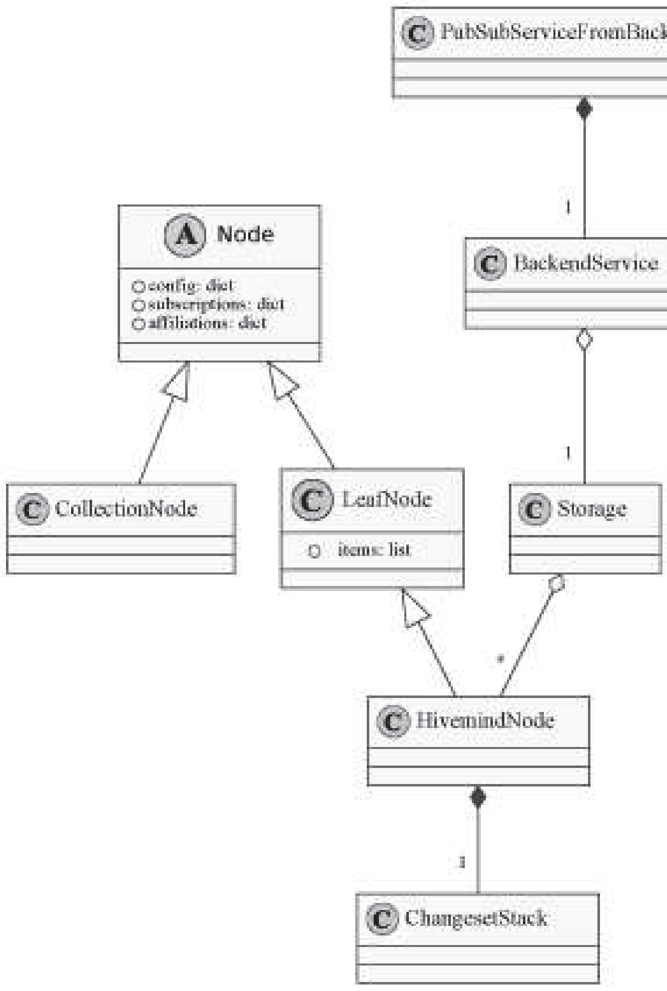


Fig. 2. Hierachy of network classes

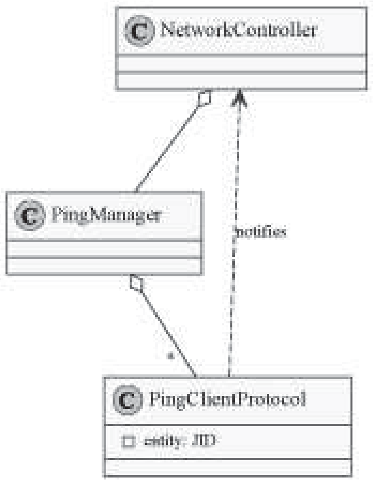


Fig. 3. PingManager interaction

Wokkel library is a pack of enhancements for Twisted. Particularly, Wokkel provides a mechanism for easier implementation of XMPP Enhancement Protocols (XEP). It supports for Service Discovery (XEP-0030), Publish-Subscribe (XEP-0060) and other XEPs. Implementation of XEP-0060 does not contain business logic. It means that Wokkel responds for receiving and generating XMPP requests according to XEP-0060 specification. Wokkel reacts on external pub/sub events and invokes corresponding methods.

XEP-0060 is very large and complicated. Implementation of all features, necessary for HiveMind, would take lots of time. Idavoll library is built on top of Wokkel and provides implementations of many XEP-0060 features such as “Subscribing”, “Publishing”, “Persistent items”, “Node creation” and etc.

Idavoll has many classes needed to implement XEP-0060 specification (see fig. 2). Node class represents a XEP-0060 node. There are two node types: Leaf and Collection. A leaf node contains a published item, whereas a collection node contains other nodes.

HivemindNode class inherits LeafNode, but stores the published items in ChangesetStack instead of a simple list. Using of ChangesetStack allows to check integrity and correctness of data. Storage class responds for managing all nodes on the service. BackendService is an implementation of XEP-0060 business logic. PubSubServiceFromBackend inherits Wokkel's PubSubService class. It invokes methods for handling corresponding XMPP requests and delegates action performing to BackendService class.

Mobile devices often have unstable internet connection, so the user have to be notified when XMPP session is lost. We check connection using XEP-0199 (XMPP Ping) [6]. PingClient-Protocol class sends ping to the XMPP server and waits for reply. If there is a certain number of failed pings, the connection is considered as lost.

The similar mechanism is used for checking participation status of a person. On the service each participant has his/her own instance of PingClientProtocol. Ping handler on the client side replies only to the service which it is connected to. We implemented PingManager class to manage multiple instances of PingClientProtocol (see fig. 3). It creates/deletes instances and starts/stops ping to the certain entity. PingClientProtocol class is responsible for notifying NetworkController about participant status, which is shown to the service owner (see fig. 4).

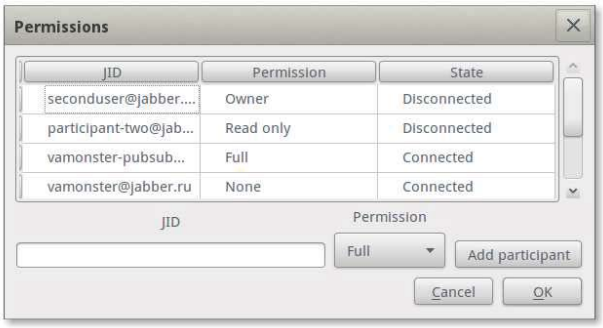
IV. ACCESS CONTROL SYSTEM

By default, all participants of teamwork have equal permissions to contribute to the mind map. It is quite convenient for brainstorming-like activities when the purpose of the collaboration is to generate some materials or to find a solution for a long-standing problem. This kind of behavior looks unsuitable for other use cases. Sometimes it is useful to restrict access for particular users in order to prevent accidental interference of teamwork. To take such scenarios into account we introduced access control system to HiveMind.

XEP-0060 provides a feature, named Access Model, which can be associated with authentication system. The user, who creates the mind map, can set trust level for all new participants i.e. control who can participate in collaboration. There are four access models implemented in HiveMind:

- "Open" — any person may join collaboration;
- "Roster" — only contacts from the owners roster may join;
- "Authorize" — the owner choose who may join on the fly. When a new person is connected, the owner gets a participation request from the person;
- "Whitelist" — a person may join only if he/she is in the owners whitelist.

"Affiliations" is the next feature, defined in XEP-0060, being a part of HiveMind access control system. This feature provides authorization capabilities to XEP-0060. After the person joined collaboration, he/she has a role, which determines a set of allowed actions. There are four roles implemented in HiveMind:


Permissions

JID	Permission	State
seconduser@jabber....	Owner	Disconnected
participant-two@jab...	Read only	Disconnected
vamonster-pubsub...	Full	Connected
vamonster@jabber.ru	None	Connected

JID

Permission: Full

Fig. 4. Permissions dialog

- “Outcast” — person not allowed to join collaboration i.e. in terms of IM he/she is banned;
- “Member” — person allowed to receive items;
- “Publisher” — person allowed to publish and receive items;
- “Owner” — similar to the Publisher, but allowed to configure collaboration behaviour.

Service owner can edit role of desired person at any time with the use of permissions dialog (fig. 4).

XEP-0060 has features that makes access control even more flexible. In some cases it might be useful to temporarily assign publisher role to all participants. HiveMind has implementation of XEP-0060 feature named Publish Model. It determines two cases of who is allowed to make changes to the mindmap. In the first case it is a person whose affiliation is publisher, and in the second case any participant is allowed to post changes.

V. Conclusion

In this paper we proposed network subsystem architecture for collaborative mind mapping based on the XMPP pubsub extension protocol. Our approach was successfully implemented in HiveMind application, which allows teamwork mode on various platforms including Maemo, MeeGo Netbook, and GNU/Linux.

HiveMind application can be downloaded from the project homepage at the following URL: *http://linuxlab.corp7.uniyar.ac.ru/projects/hivemind*. The source codes are available from the Mercurial repository at *http://linuxlab.corp7.uniyar.ac.ru/hgpub/hivemind*.

The implemented network subsystem operates in terms of abstract commands, which makes it loosely coupled from the application domain. It also makes possible to spread our teamwork architecture to a broader class of applications which could benefit from collaborative document editing.

Another direction for future development of the project is integration with SmartConference System. This kind of work is in progress now.

References

[1] A. Vasilev, A. Golovchenko, A. Kulikov, I. Paramonov “HiveMind: Cross-platform Application for Collaborative Mind Mapping,” *Proceedings of the 8th Conference of Open Innovation Framework Program FRUCT*, pp. 219–224, 2010.

[2] The XMPP Standards Foundation http://xmpp.org

[3] P. Millard, P. Saint-Andre, R. Meijer *XEP-0060: Publish-Subscribe specification.* http://xmpp.org/extensions/xep-0060.html

[4] M. Zadka, G. Lefkowitz *The Twisted Network Framework* http://twistedmatrix.com/users/glyph/ipc10/paper.html

[5] R. Meijer *Wokkel* http://wokkel.ik.nu/

[6] P. Saint-Andre *XEP-0199: XMPP Ping* http://xmpp.org/extensions/xep-0199.html