XEP-0151: Virtual Presence

Heiner Wolf
mailto:wolf@bluehands.de
xmpp:wolfspelz@jabber.bluehands.de

2005-07-05
Version 0.2

Status	Type	Short Name
Deferred	Standards Track	Not yet assigned

This document proposes extensions to the Jabber groupchat protocol for virtual presence on Web pages.

Legal

Copyright

Permissions

Warranty

Liability

Conformance

Contents

1 Introduction

Virtual presence on Web pages (also sometimes known as co-browsing, while co-browsing can also mean something different) makes people aware of each other, who are at the same Web location at the same time. The basic purpose of a virtual presence system is to show names, icons, and/or avatars of people who are on a page or a set of pages and to let them communicate. This document proposes extensions to the Jabber protocol, which enable neat virtual presence on Web pages. The extensions are implemented as namespaces for the protocol elements <iq/>, <presence/>, <message/>, and for server storage.
This document also covers the mapping of URLs to Jabber chat rooms. It describes step by step how clients derive virtual locations from URLs of web pages. The process includes queries to the web server, queries to default configuration sources, delegation between configuration sets, ways for administrators of websites to opt out, and methods to shape the virtual space actively by clustering together or splitting off URL groups.
This document describes an implementation that proved to be useful over one year. We propose the extensions, but we are open to comments and other proposals, if readers think that different approaches would better fit into the Jabber architecture. This also applies to namespaces, especially since the extensions where designed while the Jabber community was moving to URL-style namespaces. So did this implementation.

2 Requirements

2.1 Interaction on Virtual Locations

Meeting on virtual locations is very similar to meeting peers in a public chat room. But web pages can be regarded as 2 dimensional. They very often cover the entire screen. They use graphics elements for their content. Plainly speaking: representing users as figures or other images fits well to web pages. Once they are shown as individual figures, a line based chat and a chat window are not required any more (though a chat window can still be used). The figures can move around and talk in chat bubble style. Chat bubbles in turn enable incremental chat. This document describes protocol elements for:

- the visualization of users on web pages (animated avatars) and
- communication beyond line based group chat (incremental or instant bubble chat)

While users are browsing the web, they enter and leave many rooms. They meet many people and some of them multiple times. Minimum overall traffic and minimum traffic on the user connection are primary design goals. The user connection is limited by typical karma settings of jabber servers. Logging in to virtual locations (rooms) must be quick in terms of round trips and cheap in terms of traffic. Once logged in to a room, the traffic on the user connection should be independent of the number of peers already present.
The extensions have been designed to be:

- compatible to the existing Jabber infrastructure and protocols,
- lightweight with respect to the number of new elements,
- lightweight with respect to the traffic generated.

The traffic goals can be met by using only the initial <presence/> stanza, which carries all required information, so that no peer-to-peer messages are required on entering. VP clients which gather additional information about peers (e.g. avatar images) should cache the data so that it can be re-used. This is especially important since users browsing virtually connected locations (i.e. linked pages) may meet very often in a short time.
The virtual presence extensions make use of Jabber group chat. Virtual locations are implemented as public Jabber chat channels. The proposed protocol works with Multi-User Chat (XEP-0045) [1] rooms and GroupChat 1.0 compatible services. Its core functionality described here uses only groupchat features.
The virtual presence extensions are supposed to be implemented by Jabber IM clients in addition to the IM functions or by pure virtual presence (Jabber VP) clients.

2.2 Discovering Virtual Locations

The virtual presence network is a distributed network of Jabber conference components. Each component is hosting a part of the Web. Conference components may be specifically set up for virtual locations, or they may run chat rooms for virtual locations in addition to other rooms. A website may choose to run a conference component to host the presence on its web pages or it may use a publicly available service. Websites, which do not care, are covered by default servers, while others may opt out entirely in order to avoid virtual presence on their pages.
Usually VP clients gather URLs from active web browsers. They convert these URLs to Jabber chat rooms IDs (JIDs) using web server specific and general rules for the mapping process. Then they enter the room as described in the section 'Example of a Virtual Meeting'. There are many different types of URLs. Many are based on file system paths, others are query style, or a mix of path and query. Sometimes users regard individual pages as locations, sometimes a 'location' consists of multiple pages. Large websites may even consist of multiple DNS names, so that users expect to be at the same location regardless of the actual server name.
Usually website operators will not care about virtual presence on their pages. But once there is a significant amount of chat on their pages, they want to control what happens there. In other words: they need to be able to enforce domestic authority on their pages and even expel visitors from 'their' chat. They will install moderators (persons or software). Moderators authorized by the website might operate on public chat servers, but control is only guaranteed, if a website controls the assignment of moderators by running the chat server. Hence, website operators must be able to control the mapping process so that they can assign chat server addresses (and chat room names) to URLs. Many websites are hierarchical, with sub-folders being managed by different independent owners. The mapping process

[1] XEP-0045: Multi-User Chat <https://xmpp.org/extensions/xep-0045.html>.

should allow for detailed mapping rules for sub-folders and general rules closer to the root of websites, so that the 'inner' config prevails.
The mapping process is controlled by configuration files. The files will be downloaded from websites and/or from a default location. Most websites do not know about virtual presence and if they know, then they do not care. Therefore, the mapping process should be designed to allow for virtual presence without cooperation of the website. A default configuration with a simple standard mapping will be used if a website does not provide mapping rules.
Configuration files for the mapping process will often be downloaded over the network. Since the data for multiple virtual locations might add up to large files, the configuration can be split up into individual documents. The splitting is managed by delegation to other config files for subsets of URLs. This is especially important for the default configuration, which may cover special rules for prominent websites.
Global fallback servers are not able to carry all the enter/leave and chat traffic. The virtual presence of many users on many websites demands commercial services for virtual presence hosting. Much like web hosters provide disk space, traffic and processing power, there are commercial (or free) presence hosting services, which provide traffic and processing power. Website operators will either run their own presence servers or rely on commercial services. Commercial operators are limiting the ad-hoc creation of chat rooms for virtual presence or they allow static rooms only. Website operators encode the 'rented' room JID into their mapping configuration. They might even delegate the handling of configuration files entirely to the presence server operator by means of config file delegation.
The mapping process should try to protect the privacy of the user. After all, virtual presence is a violation of privacy in general, because people know where other people are (virtually). This is critical, if compared to the totally un-observed Web without virtual presence. In the real world people are used to being seen in physical locations, but only by others who are physically present in the same location. The mapping process should emulate this restriction. The general idea is to include a one way function (message digest) during the mapping process to prohibit the discovery of 'interesting' chat room names. In other words: observers must do the forward mapping and enter a room to see who is there or discover random room names without being able to re-create the source URL. So, they may be able to find people in random rooms, but do not know the virtual location.
This mapping process is designed to:

- make the virtual presence service a distributed network of jabber servers, i.e. conference components,
- allow for flexible mapping from URLs to JIDs, taking into account that
  - most URLs are path based,
  - URLs contain queries,
  - sometimes only the query part defines the virtual location,
  - groups of URLs map to a single JID,
  - a single virtual location may comprise multiple web servers,

  - groups of URLs may only cover a sub-folder of path based URLs,
  - not all URLs are known at config time,
- allow operators of websites to control the mapping for the URL-space they control,
- let websites opt out of virtual presence,
- allow for hierarchical configuration for file system path based URLs,
- support delegation,
- allow for virtual presence without the cooperation of the website,
- allow for distribution of the load of the default configuration server,
- support commercial virtual presence servers and rented rooms,
- be extensible to other protocols as virtual presence transport,
- be easily implemented by website operators,
- at least limit the privacy issues associated with virtual presence.

3 Example of a Virtual Meeting

Since it has been suggested that the use of Shakespeare characters might please the reader we will try to employ them without really knowing the works of Shakespeare. Two characters called Romeo and Juliet will appear. Both will be equipped with a Web browser and a virtual presence Jabber client which implements the protocol extensions described in this document (called VP client in the following). The communication is shown as seen by the conference component including 'from' attributes.

3.1 Meeting on Virtual Locations

This shows how the first meeting of Romeo and Juliet would have happened if they had known Cyberspace, i.e. the Web. Romeo browses the Web with his favorite web client. He reads the page http://www.shakespeare.com/market/ModernLibrary/page1.html. He is also running a VP client. There is no mention of a buddy list jabber client at this point. The VP client is properly logged in to the Jabber server using the JID romeo@montague.net. Through some magic, the VP client acquires the URL of the current web page. The VP client converts the URL into the JID of a public chat room (see section 'Virtual Locations'). The JID will usually be a SHA1 digest of the URL prefixed to a conference component (like f4eb29410ec339144633773dda1dc4a643513933@shakespeare.com). For the sake of readability we will refer to the chat room as room1@shakespeare.com. The VP client then enters the room using 'YoungHero' as the nickname.

Listing 1: Entering a virtual location

```
<presence from='romeo@montague.net/garden' to='room1@shakespeare.com/
    YoungHero' />
```

The room responds:

Listing 2: Acknowledgement of the entry

```
<presence from='room1@shakespeare.com/YoungHero' to='romeo@montague.
    net/garden' />
```

Upon reception the VP client visualizes the presence of Romeo at the location to Romeo, e.g. by displaying Romeo's icon and/or nickname on, at, or close to the web page. In other words: the VP client shows an avatar of Romeo on the page.
Dramatic increases when Juliet browses to the same web page. Her VP client also gathers the URL, performs the same mapping and enters the room. It sends a <presence/> stanza to the room using nickname 'WebGirl'. The room as expected returns the acknowledgement. In addition the room forwards both <presence/> stanzas to the VP clients in order to announce the mutual presence.
Romeo gets:

Listing 3: Announcement of peer presence

```
<presence from='room1@shakespeare.com/WebGirl' to='romeo@montague.net/
    garden' />
```

Juliet gets the equivalent. Romeo's VP client shows Juliet's presence by adding a second avatar to the page. They now engage in a vivid conversation. In deviation to the original text Juliet would not ask "Who art thou, Romeo?", because she only knows the nickname and on the other hand she actually knows where Romeo is: he is on the same web page. There is nothing new until now. Everything happened according to GroupChat 1.0 (XEP-0045) without MUC features. Of course, users can join the same room with any client that supports GroupChat, if they know the room's JID.
Extensions come into the play in order to make the virtual presence more attractive and more vivid.

3.2 Getting more information

For more information about 'YoungHero' beyond the nickname Juliet needs a JID (see below for anonymous variants of the avatar image). Non-anonymous rooms will supply the JIDs automatically. But we suggest that rooms, which make up the virtual presence network are configured to be anonymous so that users can choose if they want to disclose the JID. We propose an extension to the <presence/> stanza for users to supply their JID automatically to peers in anonymous rooms with minimum traffic even for many participants.
Note: even though Romeo sends a JID, the systems is still anonymous. Romeo could send

any JID. He may send a (fake) JID that is just the base address of his storage, but not his communication address. In anonymous rooms Juliet will use any JID Romeo provides. If the room is not anonymous, then Romeo's client may use Juliet's actual JID.
Entering the room Juliet would add a JID-element to the initial <presence/> stanza.

Listing 4: Disclosing the JID

```
<presence from='juliet@capulet.com/balcony' to='room1@shakespeare.com/
    WebGirl'
  <x xmlns='firebat:user:jid'>juliet@capulet.com/balcony</x>
</presence>
```

(Note: 'firebat' was the developer name of the VP client. Jabber.org URIs could be used here)
This tells the conference component to send the JID to all participants automatically. Romeo will receive the <presence/> stanza including the JID element. Romeo may now fetch Juliet's avatar or add Juliet to a buddy list.
Note: disclosing the JID is usually not advisable in public rooms. We decided to offer the functionality as an option, for 2 reasons:

1. to provide access to extended information from peers without cluttering the <presence/> stanza more than necessary,
2. to allow for caching of extended information.

Caching requires a persistent and unique id per user. While a message digest of the JID would be sufficient for caching extended information, it is not sufficient for retrieving extended information.

3.3 Avatars

Romeo will be much more interested in Juliet if Juliet is depicted by a nice image rather than only a name. Users may choose avatars, publish those avatars, and change them on the fly. Changes to the avatar are tracked by comparing a stored digest of the avatar data to the received digest. The avatar digest will be received as part of the <presence/> stanza. Juliet also includes an avatar-digest element with a hex-coded SHA1-digest of the avatar data (i.e. the digest of the image file):

Listing 5: Avatar Digest

```
<presence from='juliet@capulet.com/balcony' to='room1@shakespeare.com/
    WebGirl'
  <x xmlns='firebat:user:jid'>juliet@capulet.com/balcony</x>
  <x xmlns='firebat:avatar:digest'>9
     b3635eb1440a1ee1c9f67767651194f34ecf130</x>
</presence>
```

Romeo receives the digest with the initial <presence/> stanza from Juliet. Since this is their first meeting Romeo does not have Juliet's avatar yet. He will fetch the avatar from Juliet's public XML server storage (for a public XML storage free variant see below):

Listing 6: Avatar Request

```
<iq type='get' to='juliet@capulet.com'>
  <query xmlns='storage:client:avatar' />
</iq>
```

The server returns Juliet's avatar.

Listing 7: Avatar Response with Avatar URL

```
<iq type='result' to='romeo@montague.net/garden'>
  <query xmlns='storage:client:avatar'>
    <data src='http://www.capulet.com/juliet.png'/>
  </query>
</iq>
```

The 'src' attribute points to the real data, which is to be fetched out of band via HTTP. Juliet supplied a reference to the data because she is a smart WebGirl concerned about bandwidth on the Jabber connection. But if the Capulets would not own a domain she could also choose to store the avatar data directly on the server using base-64 encoding. The avatar response would then be like:

Listing 8: Avatar Response with Immediate Data

```
<iq type='result' to='romeo@montague.net/garden'>
  <query xmlns='storage:client:avatar'>
    <data mimetype='image/png' encoding='base64'
      >E00fcB79F822u192e7A2067E5229ec136892A296... </data>
  </query>
</iq>
```

Juliet can omit the 'mimetype' attribute if she supplies the avatar URL, because the HTTP server will return the MIME type. The 'mimetype' attribute must be included for immediate data. Avatars may be of any type and size. The maximum visible dimensions of avatars should be 64x96 pixels. Larger images will be scaled down to fit into this rectangle. They will not be scaled up. Images align at the bottom so that they can 'stand' on a page on the same base line.
Implementation note: The client should check the image size (byte count) and advice the user if the size exceeds 20 kb. It should also check the image dimensions and warn the user if the image will be scaled down. Checking the size is especially important for immediate data, which will be uploaded over a Jabber connection which is subject to flow control. Typical karma settings only allow for few kb without stalling the connection.
Implementation note: Avatars and associated digests should be stored permanently. They will always be up to date as soon as people meet and exchange digests. There is no cache

timeout. The implementation should keep an eye on the avatar storage and delete old and rarely used ones.
Avatars images can also be created of other data types. The following shows an avatar description in XML format, which is to be processed by an advanced avatar engine. The avatar engine will be chosen by the MIME type. For simple image based avatars, the MIME type selects the type of image decoder. Other MIME types may select from a dynamic list of installable avatar rendering plug-ins.

Listing 9: Avatar Response for an Advanced Avatar

```
<iq type='result' to='romeo@montague.net/garden'>
  <query xmlns='storage:client:avatar'>
    <data mimetype='avatar/comic' encoding='plain'>
      <config xmlns='http://schema.bluehands.de/character-config'
          version='1.0'>
        <character name='lluna' version='1.0'
         src='http://avatar.vp.bluehands.de/comic/lluna.xml'/>
        <color rgb='#00FF00' name='head'/>
        <color rgb='#999900' name='shadow'/>
        <color rgb='#000000' name='border'/>
        <color rgb='#FFFFFF' name='tooth'/>
        <color rgb='#FFFFFF' name='eye'/>
        <color rgb='#000000' name='pupil'/>
        <color rgb='#000000' name='mouth'/>
        <color rgb='#FF0000' name='tail1'/>
        <color rgb='#FF0000' name='tail2'/>
        <color rgb='#FF0000' name='tail3'/>
        <scale ratio='0.8'/>
      </config>
    </data>
  </query>
</iq>
```

The example avatar data above will be interpreted by the avatar engine, which registered for the MIME type 'avatar/comic'. This is a sample implementation of an animated avatar. The data states that an animation file shall be loaded from http://avatar.vp.bluehands.de/comic/lluna.xml and then the character config shall be applied modifying the colors of the character definition.
The design is particular targeted at existing online games as avatar providers. It would be very easy to adapt the avatar rendering of an online role playing game (MMORPG) or any other community so that it provides avatar images for the VP client. In this case the avatar data would probably only consist of the online accout ID. This is up to the implementation. We imagine such avatar data to be like:

Listing 10: Potential Role Play-Plugin Avatar Data

```
<iq type='result' to='romeo@montague.net/garden'>
```

```
  <query xmlns='storage:client:avatar'>
    <data mimetype='rpg/my-favorite-mmorpg' encoding='plain'>ACCOUNTID
       =6575438998</data>
  </query>
</iq>
```

... and the plug-in (which is part or the game distribution) connects to the game server, fetches the configuration, and renders Juliet as an nightelve mage level 30 with all insignia on the web page of Romeo. Romeo is very impressed, falls in love instantly and they both die in the course of a tragic story. Provided that Romeo happens to have the same gaming engine installed. Having not might save him in this time.
Note: As a matter of fact, we propose a call level plug-in interface formerly described at http://developer.lluna.de/docs/animated-avatars.html for advanced avatars.
Note: To make things simple for the implementation users may have 2 avatars. The first avatar ('storage:client:avatar') is restricted to GIF and PNG images. The second avatar is fully featured, but optional. It is accessed through the namespace 'storage:client:avatar2' and tracked through digest 'firebat:avatar2:digest'. VP clients must implement 'storage:client:avatar' and 'firebat:avatar:digest' with at least PNG support. They may implement 'storage:client:avatar2' and 'firebat:avatar2:digest'. The scheme can be extended with higher numbers.

3.4 Other Ways to get the Avatar

If Juliet wants to protect her real JID, then she can not use public XML server storage. She can still provide an avatar by including the avatar data into the <presence/> stanza.

Listing 11: Explicit Avatar Presence Data

```
<x xmlns='firebat:avatar:data'><data src='http://www.capulet.com/
   juliet.png'/></x>

as part of:
<presence from='juliet@capulet.com/balcony' to='room1@shakespeare.com/
   WebGirl'
  <x xmlns='firebat:avatar:digest'>9
     b3635eb1440a1ee1c9f67767651194f34ecf130</x>
  <x xmlns='firebat:avatar:data'><data src='http://www.capulet.com/
     juliet.png'/></x>
</presence>
```

Juliet must not send bulky immediate data, but a reference to the image. Immediate data is only allowed, if it is smaller than a simple <presence/> stanza (see above).

Listing 12: Explicit and Immediate Avatar Presence Data

```
<x xmlns='firebat:avatar:data'>
```

```
  <data mimetype='rpg/my-favorite-mmorpg' encoding='plain'>ACCOUNTID
     =6575438998</data>
</x>
```

Note: There are other features which won't work without the real JID. Therefore, it is recommended that clients include the JID in the <presence/> stanza. If Juliet wants to use all JID-based features then she would rather use a public Jabber server with an anonymous account, like she is used to do in case of email. This feature is supported because there are still Juliets who refrain from disclosing the JID and because the avatar is very important.
If Juliet provides the JID, but public XML server storage is not supported on Juliet's server, then Romeo may try to discover the avatar using disco features. He will query Juliet's server avatar disco node

Listing 13: Avatar Discovery Request

```
<iq to='juliet@capulet.com' type='get'>
  <query xmlns='http://jabber.org/protocol/disco#info' node='http://
     jabber.org/protocol/avatar'/>
</iq>
```

The result contains an item with the JID of the pubsub component where Juliet's avatar information is published and the specific node for that information:

Listing 14: Avatar Discovery Response

```
<iq type='result' from='juliet@capulet.com' to='romeo@montague.net/
    orchard' >
  <query xmlns='http://jabber.org/protocol/disco#items'
    node='http://jabber.org/protocol/avatar'>
    <item jid='pubsub.shakespeare.lit' node='avatar/info/
       juliet@capulet.com'/>
  </query>
</iq>
```

3.5 Avatar Movement

A web page can be regarded as a two dimensional space. Avatars can move around on the page. Users are not just at a page, they are at a certain coordinate of the page. Entering the room Juliet tells Romeo (and all other participants) her initial position by including a position element into the initial <presence/> stanza:

Listing 15: Avatar Position

```
<presence from='juliet@capulet.com/balcony' to='room1@shakespeare.com/
    WebGirl'
  <x xmlns='firebat:user:jid'>juliet@capulet.com/balcony</x>
```

```
  <x xmlns='firebat:avatar:digest'>9
      b3635eb1440a1ee1c9f67767651194f34ecf130</x>
  <x xmlns='firebat:avatar:position'><position x='180' w='853'/></x>
</presence>
```

The coordinates are in pixels from the bottom left corner of the browser window. In this case she only tells the horizontal position. She could add a 'y' attribute. She also voluntarily discloses the width of her browser window. The width ('w') and height ('h') attributes are optional. They enable relative positioning at the other end.
As soon as Juliet enters and her avatar appears, Romeo moves his avatar up to Juliet's. Romeo's VP client sends a new <presence/> stanza with an altered position. The message is automatically broadcast by the room to all participants and stored for new participants. Romeo approaches Juliet:

Listing 16: Move Avatar

```
<presence from='romeo@montague.net/garden' to='room1@shakespeare.com/
    YoungHero'
  <x xmlns='firebat:user:jid'>romeo@montague.net</x>
  <x xmlns='firebat:avatar:digest'>
      e5229ec136892e00fcb79f822c192e7a2067a296</x>
  <x xmlns='firebat:avatar:position'><position x='150'/></x>
</presence>
```

Note that this <presence/> stanza, which is sent in order to move the avatar is not different from the initial <presence/> stanza to the room. This is just a repetition with a different coordinate.
Juliet's VP client forwards the movement to the avatar rendering engine. In case of an image based avatar the image is just moved to take up the new position. In case on an animated avatar, the avatar engine will generate a sequence of images which lets the figure 'walk' to the new position or whatever movement method is appropriate for the avatar.
Now Romeo occupies exactly the part of the web page that Juliet is reading so she moves Romeo (locally) out of the way. This happens only on Juliet's screen. Once Juliet finished reading, she wants to restore Romeo's position. She sends a control message to Romeo to request the current position:

Listing 17: Request Avatar Position

```
<iq from='room1@shakespeare.com/WebGirl' to='room1@shakespeare.com/
    YoungHero' type='get'>
  <query xmlns='http://schema.bluehands.de/avatar#position'/>
</iq>
```

Romeo responds with a control message which indicates the avatar position:

Listing 18: Send Avatar Position

```
<iq from='room1@shakespeare.com/YoungHero' to='room1@shakespeare.com/
    WebGirl' type='result'>
  <query xmlns='http://schema.bluehands.de/avatar#position'>
    <position x='123' w='800'/>
  </query>
</iq>
```

Juliet' VP client restores Romeo's avatar position:

3.6 Bubble Chat

We remember: after seeing Juliet's avatar, Romeo has already falling in love. But Juliet is a more realistic type. She insists on talking before falling in love. The avatar-on-web-pages paradigm frees us from chat line based chat. Each avatar may show a separate chat bubble with the latest text or even a chat history.
While Juliet is writing an opening, her VP client may send the current contents of the chat line to the room giving other participants snapshots of the text. We propose instant chat updates as an extension to groupchat. For backward compatibility these snapshots are transmitted as control messages without body text, so that other clients ignore them. Once Juliet hits the send button (or the enter-key) the VP client sends the chat message as body text of a <message/>. With a timely distance of fractions of seconds to few seconds Juliet sends the following messages:

Listing 19: Instant Chat

```
<message from='room1@shakespeare.com/WebGirl' to='room1@shakespeare.
    com'>
  <x xmlns='firebat:chat:state'>Who</x>
</message>

<message from='room1@shakespeare.com/WebGirl' to='room1@shakespeare.
    com'>
  <x xmlns='firebat:chat:state'>Who art thou,</x>
</message>

<message from='room1@shakespeare.com/WebGirl' to='room1@shakespeare.
    com'>
  <x xmlns='firebat:chat:state'>Who art thou, YoungHer</x>
</message>

<message from='room1@shakespeare.com/WebGirl' to='room1@shakespeare.
    com'>
  <body>Who art thou, YoungHero?</body>
</message>
```

Note: we propose to send the full text with every message. Differential updates seem to be the proper way. But the addressing information sent with each message produces more traffic

than the enclosed text. This is especially true for randomly chosen (read: long) room names. On the other hand differential text requires synchronization points for new participants, who lack the history. Such synchronization points would carry the full text every other second rendering differential updates almost useless.
Implementation note: The traffic must stay below the karma-limitations of typical jabber servers. The minimum delay between instant chat messages should be 1 sec. The delay should be increased with the size of the text. We get a good balance between interactivity and traffic with the following algorithm (in msec): delay = max(5000, min(NumberOfChars x 25, 1000)). In words: from 1 second at 40 characters up to 5 seconds at 200 characters.
Implementation note: There is no limitation on the size of the instant chat specified. But some users, especially the ones not used to chat channels write without ever pressing the enter-key. Implementations should limit the size to reasonable length (200-1000 characters) by telling the user, refusing to send more, or just stripping the beginning of a long text.

3.7 Video Icon

After some chatting, Juliet is still not convinced. She demands to see Romeo live (although not in person). Romeo switches on his webcam and sends the webcam URL as part of the <presence/> stanza.

Listing 20: Iconic Video

```
<presence from='romeo@montague.net/garden' to='room1@shakespeare.com/
    YoungHero'
  <x xmlns='firebat:user:jid'>romeo@montague.net</x>
  <x xmlns='firebat:avatar:digest'>
      e5229ec136892e00fcb79f822c192e7a2067a296</x>
  <x xmlns='firebat:avatar:position'><position x='150'/></x>
  <x xmlns='firebat:icon:video'>http://romeo.montague.net/video/icon</
      x>
</presence>
```

Juliet's VP client HTTP-requests the video stream. The video is shown close to Romeo's avatar. Juliet falls in love.
The video format is the common, widely used webcam format introduced by Netscape (JPEG server-pushed).
In the real world there are many Romeos and Juliets even on the same page at the same time. Some of them with a webcam. Depending on the default settings of VP clients all users (say 8) will automatically request the videos of the subset of users equipped with a webcam (say 3). This automatically creates many video streams, 3 streams on the dialup line of all users and 7 streams on the line of webcam users. We therefore suggest the following optimizations and limitations:

- video dimensions should be limited to 64x64 pixels, thus fitting into the 64x96 avatar

rectangle,

- the frame rate should not exceed 3 frames per second for simple web browsing (special applications, especially assymmetric ones, like virtual class rooms, presentations may differ),
- for small sizes, quantization tables and Huffman tables make much more data than the encoded image. Therefore, the DQT and DHT markers should be stripped from JPEG frames except for the first frame of a stream. Only the JPEG baseline algorithm is supported with static Huffman tables,
- VP clients which support such a optimized JPEG server-push format should add a 'Accept: image/djpeg' header to the HTTP request. (djpeg for differential JPEG)

The purpose of these limitations is to allow for webcams which send optimized streams of small images (reducing the data volume by a factor of 3) while supporting usual webcams.

4 Finding Virtual Locations

Finding a virtual location for a websites means mapping a URL to a JID. The VP client gets the URL from the browser by some platform dependent way of interprocess communication. The URL may have been entered by the user, or it may be the result of a URI resolver. Then the VP client searches for an applicable mapping rule. This may include fetching multiple configuration files over the network. Finally the rule is applied and the VP client can enter the room.
The mapping process is explained in great detail here. Server admins only need the sections 'Mapping Rules' and 'Config Files'

4.1 Mapping Rules

The basic mechanism is very simple: a URL is mapped by means of a regular expression with replacement term. A hash function may follow after the regular expression and an optional prefix can be added for convenience. The mapping rule produces a room name and a chat service URL. The chat service URL points to the server which hosts the channel. It also tells which protocol to use. Protocol, server address and channel name identify the chat channel.
Romeo is on http://www.shakespeare.com/market/ModernLibrary/index.html. A single mapping rule might cover the entire URL space of http://www.shakespeare.com/. It might look like:

Listing 21: Basic Mapping Rule

```
<location match='^http://www\.shakespeare\.com/([^/]+)/.*$'>
    <name>\1-room</name>
```

```
    <digest><prefix>vp-</prefix></digest>
    <service>xmpp:conference.shakespeare.com</service>
  </location>
```

Listing 22: Basic Mapping Rule Result: a Room JID

```
vp-85b0df53e7ce7d2e0406d2bbf8a9d699aaa9db53@conference.shakespeare.com
```

This <name/>-tag produces the room name. The text of the <name/>-tag in combination with the match expression of the <location/> extracts the first level folder of the path section from the URL. It creates a room for each first level folder. The resulting room name is 'market-room'. It is hashed by a message digest, if the <digest/> tag is present. A SHA1 digest is used by default. VP clients must support SHA1. The hashed result will be prefixed by 'vp-'. Prefixing is provided for convenience, so that rooms generated for virtual presence can be sorted easily in room lists of conference components. Both <digest/> and <prefix/> are optional.
The \1 variable of the match-expression is used to generate a room per sub-folder of the URL. Match-expression variables can be used inside all elements to modify the inner text of the element. You might use the \1 inside the <service/>-tag to point to a different chat server per sub-folder of the URL.
The rule generates a virtual location ID. This mechanism is extensible to other protocols as virtual presence transport, e.g. IRC. The <service/>-tag contains a service URL, which consists of a scheme and server address. In this case the 'xmpp'-scheme means that the Jabber protocol will be used and that the VP client joins a Jabber chat room with a JID created according to the Jabber ID building rules, i.e. room-name@server-address, where the room-name has been derived from the 'name'-tag and mangled by the 'digest'-tag before it has been prepended to '@server-address'.
A <location/> without match-attribute matches all URLs. In other words: the default match-attribute is '.*'.

4.2 Config Files

Rules can be stored in various ways. If they are stored in files, then the rules are wrapped by a toplevel element. The VPI toplevel element can contain multiple <location/>-elements (rules):

Listing 23: VPI file format

```
<?xml version='1.0' ?>
<vpi xmlns='http://schema.bluehands.de/virtual-presence-info'>
  <location match='...'>...</location>
  <location match='...'>...</location>
  ...                          <!-{}-  more locations  -{}->
  <location>...</location>     <!-{}-  default location -{}->
</vpi>
```

Examples were formerly available at http://developer.lluna.de/docs/vpi-file-syntax.html

4.3 Extensions to Mapping Rules

Independent services may publish lists of virtual locations. An example is a 'topsites'-service, which publishes a list of most active locations as a list of URLs. These services usually require the cooperation of VP clients. VP clients publish their locations (and URLs), if configured to do so. In some cases website operators allow for virtual presence on their pages, but they do not want that the real URLs are published, or they do not want the URLs published at all. Therefore, the location configuration offers optional tags to control how URLs are disclosed by VP clients. A <hidden/>-tag prohibits the publication of the URL. A <destination/>-tag modifies the URL. A <hidden/> location may look like:

Listing 24: Hidden Location

```
<location match='^http://www\.shakespeare\.com'>
    <name>shakespeare</name>
    <digest/>
    <service>xmpp:conference.shakespeare.com</service>
    <hidden/>
  </location>
```

In this example all URLs of the website will be mapped to a single room. Users will be present there, but the room will not appear in public lists (if clients and publication services comply). A simplification of the above is the <mapped/>-tag, which does not employ regular expressions. All URLs of the website can be mapped to a single location (omitting the <hidden/> feature again):
The <destination/>-tag modifies the URL so that website operators control which URLs are published. They might delete a web-session ID from URLs or alter the URL so that users who discover the location on a publication service are directed to a different URL. The biggest concern is that the exposed URL may contain security relevant session data. The <destination/>-(per <rule/>) allows to remove this data from URLs while preserving browseable URLs. The following example deletes any path and query from URLs.

Listing 25: Destination: Modifying Published URLs

```
<location match='^http://www\.shakespeare\.com'>
    <name>shakespeare</name>
    <digest/>
    <service>xmpp:conference.shakespeare.com</service>
    <destination>http://www.shakespeare.com/</destination>
  </location>
```

Note: VP clients should not rely on a <destination/> only, when publishing URLs. A client should consult the user if, how, and where URLs are to be published. It is recommended to

strip the query part by default until configured otherwise.
The <ignore/>-tag prevents virtual presence clients from dealing with matching URLs. There will be no virtual presence associated with the URL.

Listing 26: Ignore: Opt out

```
<location match="\path{^http://www\.shakespeare\.com($|/.*$)}">
  <ignore/>
</location>
```

4.4 Web Server based Configuration

Location configuration data, such as the above is stored in configuration files called Virtual Presence Information (VPI) files. The files are fetched via HTTP or FTP, according to the scheme of the URL which is to be mapped. Websites can offer VPI files at each level in the path hierarchy. If they do not offer VPI files then the VP client reverts to default VPI. The VP client tries to find a VPI file in the same folder as the file of the URL. It strips the file and query parts of the URL and appends the file name _vpi.xml:

Listing 27: Nearest VPI File

```
http://www.shakespeare.com/market/ModernLibrary/_vpi.xml
```

HTTP-servers have many different ways to respond if the file is not available, which is the most common case. They should respond with a 404 status code and an optional HTML message. But some web servers are configured to return a default document, some return a status code 200 and a 'Content-type: text/html' message with an error text, some even return status code 200 and 'Content-type: text/xml' with an HTML message. It is therefore recommended that VP clients check the HTTP status code, the Content-type and the validity of the XML document. If there is no such file or if it is not a valid XML file or, in case of HTTP, if the Content-type is not text/xml, then the request fails. If the request fails then the client ascends the path until it hits the top level of the website trying to fetch:

Listing 28: Web Site Toplevel VPI File

```
http://www.shakespeare.com/_vpi.xml
```

If the request fails, then the client reverts to a default VPI file if there is any. The current implementation uses`http://vpi.vp.bluehands.de/lluna-2.5.2/root-vpi.xml`. It will finally use http://www.virtual-presence.org/LMS/root-vpi.xml (LMS = Location Mapping System) or another global configuration file. http://vp.jabber.org/root.xml would be an option.
If there is no default VPI file then the VP client uses the following configuration data:

```
<location>
  <name>\1</name>
```

```
  <digest/>
  <service>xmpp:location.virtual-presence.org</service>
</location>
```

This configuration creates a virtual location for each web server address. The server name is obfuscated by a hash function (SHA1). The location configuration applies to all URLs, since the match-attribute of the <location/> defaults to '.*'.
VP clients must cache the result of VPI downloads including failed requests and invalid responses. They must keep a local copy for at least a few minutes to avoid traffic from multiple (ascending) VPI file downloads.
Note: There are many websites, which consist of multiple web server addresses. This is especially true for large sites, which use DNS round-robin for load balancing. These websites need a special mapping to meet the expectation of the user, because users usually do not care if they are on www01.website.com or on www02.website.com. If users expect to see each other, then the virtual presence service should meet the expectations. There might also be website operators, who do not like virtual presence on their pages. These operators may even take legal action against the provider of the VP client software. So, these websites may require special configuration as well. And there may be other reasons for individual configuration of selected websites. Therefore, it is recommended, that VP clients update their default configuration, i.e. by using an automatic update feature or by downloading the default VPI.
A VPI file may delegate the handling of URL spaces to another VPI file. The <delegate/>-node replaces the <mapping/> and <service/> nodes of the <location/>. If the VP client discovers a <delegate/> tag inside a <location/>, which matches the URL to be mapped, then it stops processing the file and loads the VPI file denoted by the <delegate/>-tag. A request to http://www.shakespeare.com/market/ModernLibrary/_vpi.xml could return VPI like:

Listing 29: Delegation

```
<location>
  <delegate>http://www.shakespeare.com/_vpi.xml</delegate>
</location>
```

This means that the VP client skips`http://www.shakespeare.com/market/\_vpi.xml` and continues with http://www.shakespeare.com/_vpi.xml. Delegation can be used to direct the VP client to the VPI file of a commercial virtual presence service. The commercial service would then be responsible to create rooms for customers and to provide the appropriate VPI. Delegation is also useful to split up the default VPI file. The default VPI file contains special configuration for some large websites. It has been split into individual files for different (DNS) top level domains. The global VPI file currently in use (http://vpi.vp.bluehands.de/lluna-2.5.2/root-vpi.xml) contains the following statement for the .com space:

Listing 30: Delegation for Subspaces

```
...
```

```
<location match="\path{^http://.*\.com($|/.*)}">
  <delegate>http://vpi.vp.bluehands.de/lluna-2.5.2/dotcom-vpi.xml</
     delegate>
</location>
...
```

4.5 The Mapping Process

This section is intended as a guideline for the implementation of the mapping process.
The mapping has 2 phases:

1. finding the rule
2. applying the rule

4.5.1 Find the Rule

We get a URL from the web browser, say:

```
http://www.shakespeare.com/market/ModernLibrary/index.html
```

We try to fetch the configuration file from

```
http://www.shakespeare.com/market/ModernLibrary/_vpi.xml
```

The request fails (the failure is noted in the cache), and we try

```
http://www.shakespeare.com/market/_vpi.xml
```

The request fails again (the failure is noted in the cache). We try

```
http://www.shakespeare.com/_vpi.xml
```

We do not find it (the failure is noted in the cache), and we revert to the global file (which one depends on the client configuration)

```
http://vpi.vp.bluehands.de/lluna-2.5.2/root-vpi.xml
```

The request returns (and the data is stored in the cache):

```
<?xml version='1.0' ?>
<vpi xmlns='http://schema.bluehands.de/virtual-presence-info'>
```

```
  <!-- handle .com(s) by another file -->
  <location match='^http://.*\.com/.*'>
    <delegate>http://vpi.vp.bluehands.de/lluna-2.5.2/dotcom-vpi.xml</
       delegate>
  </location>

  <!-- handle .de(s) by another file -->
  <location match='^http://.*\.de/.*'>
    <delegate>http://vpi.vp.bluehands.de/lluna-2.5.2/dotde-vpi.xml</
       delegate>
  </location>

  <!-- all others go here -->
  <location match='^http://([^/]+)($|/.*$)'>
    <service>xmpp:location.virtual-presence.org</service>
      <!-- one location per website  -->
      <name>\1</name>
      <digest/>
  </location>
</vpi>
```

We try to find a <location/> that matches our URL. We find:

```
<location match='^http://.*\.com/.*'>
  <delegate>http://vpi.vp.bluehands.de/lluna-2.5.2/dotcom-vpi.xml</
     delegate>
</location>
```

This means that all .com domains are forwarded to a separate VPI file. We fetch:

```
http://vpi.vp.bluehands.de/lluna-2.5.2/dotcom-vpi.xml
```

We store the result in the cache and search for a matching <location/> again. We find the default section (rest of the file omitted, the match-attribute is more general than the one of the previous <delegate/>, because here we are already in the .com domain):

```
...
<location match='^http://([^/]+)($|/.*$)'>
  <service>xmpp:location.virtual-presence.org</service>
  <mapping>
  <name>\1</name>
  <digest/>
</location>
...
```

The <location/> matches, so we get a virtual presence service address and a set of rules. The virtual presence server is

```
location.virtual-presence.org
```

The protocol to use is

```
xmpp
```

There mapping rule is:

```
match='^http://([^/]+)($|/.*$)'
  <name>\1</name>
  <digest/>
```

The result of the first phase is a mapping rule, which will be applied to all URLs *in the same folder* as the original URL. In regex-speech:

```
http://www.shakespeare.com/market/ModernLibrary/.*
```

To apply the mask only to URLs in the same URL-path folder is a security requirement, so that 'inner' VPI files from websites can not configure the mapping of 'outer' folders or 'siblings'.
The original URL was:

```
http://www.shakespeare.com/market/ModernLibrary/index.html
```

So, the security mask is (the rule applies only to URLs in the same folder as the original URL):

```
^http://www\.shakespeare\.com/market/ModernLibrary/.*
```

If the security mask applies to a URL can be verified by a simple string-compare without using a regular expression.
The <location/> has a match attribute. This is the user supplied mask of the rule:

```
^http://([^/]+)($|/.*$)
```

4.5.2 Apply the Rule

The URL where we want to meet people is:

```
http://www.shakespeare.com/market/ModernLibrary/index.html
```

We got many rules with a security mask (from the folder of the original URL) and a regular expression (from the match-attribute). For each new URl we check the URL against the security mask and the regular expression. One of the rules applies to the URL:

```
^http://www\.shakespeare\.com/market/ModernLibrary/.*
^http://([^/]+)($|/.*$)
```

The regular expression

```
^http://([^/]+)($|/.*$)
```

and the room <name>

```
\1-room
```

will extract the first level folder from the URL. The URL

```
http://www.shakespeare.com/market/ModernLibrary/index.html
```

gives:

```
market-room
```

The <digest/>-tag tells us to hash the regex replacement result with the default message digest SHA1. So we get:

```
87d0c3e8d08f344375f22014c7cafe6527acbae3
```

The <prefix/>-tag tells us to prefix with

```
vp-
```

and finally the ID of the virtual location (the room name) is:

```
vp-87d0c3e8d08f344375f22014c7cafe6527acbae3
```

From the <service/>

```
xmpp:location.virtual-presence.org
```

the client knows the transport protocol and the address. The Jabber protocol will be used as transport protocol and the Jabber conference room JID is

```
vp-87d0c3e8d08f344375f22014c7cafe6527acbae3@location.virtual-presence.
    org
```

5 Error Codes

This document does not introduce new error codes.

6 Security Considerations

The system has been designed to protect the privacy of the user as good as possible. If users decide to run the VP client, then other users may see their movement, but only if they enter the same locations. There is no way to track users, especially since the JIDs of virtual locations (Jabber chat rooms) are supposed to be SHA1 digests of URLs which are not predicable. So people trying to track Romeo have to guess the URL and enter the respective chat room to find Romeo. They cannot browse the room list of a conference component and deduce the URL from the room name.
The fact that clients may disclose their JID in order to provide access to their avatar server storage and in order to enable caching is a weak point. A different but equally unique ID could be for caching purposes, provided that it is part of the <presence/> stanza. But traffic restrictions on the client connection prevent that additional data is exchanged between peers in a room. The data could be broadcast to the room, but this could end up in many people always broadcasting avatars while entering a room (web page). The chosen solution is to let peers download from public server storage. A better solution might be to let the conference component fetch from the server storage. In this case clients would need only the room-JID of the user, not the real JID.

7 XMPP Registrar Considerations

These namespaces need to be reviewed and/or registered with the XMPP Registrar as a result of this document:

- firebat:user:jid
- firebat:avatar:position
- firebat:avatar:getpos
- firebat:chat:state
- firebat:icon:video
- firebat:avatar:digest
- firebat:avatar2:digest
- storage:client:avatar

- storage:client:avatar2

8 Formal Definition

8.1 Schema

No schema definitions yet. They will be added in the next version of this document, if it is considered for publication.

9 Conclusion

The virtual presence on Jabber has been designed to fit easily into the existing Jabber infrastructure including existing software components, clients, and protocols. It turns out that Jabber offers everything necessary for basic virtual presence.
This document proposes a mapping process in order to create a space for virtual presence on top of the URL based Web infrastructure. It also proposes namespace extensions for the protocol, which make virtual presence on web pages more convenient. The core features are:

- URL mapping and service discovery,
- avatars standing and walking on a web page,
- bubble chat,
- iconic video.

There are definitely more features possible. Suggestions are welcome