Communications instantanées pour le portail Silverpeas

Rapport de stage final,

Master IC2A DCISS RS 2015/2016,

Rémi Pace

Table des matières

1. Introduction

L'objet de ce stage est le renouvellement de la messagerie instantanée du portail collaboratif Silverpeas. L'ensemble des travaux présentés ci-dessous est disponible en ligne:

- Espace de démonstration du système complet:
  https://im.silverpeas.net
- Système intégré au portail Silverpeas :
  https://im.silverpeas.net/silverpeas-intro.html
- Client JSXC original :
  https://im.silverpeas.net/jsxc.original/example/

```
XMPP domain : im.silverpeas.net
BOSH url : https://im.silverpeas.net/http-bind/
AJAX Login : Identifiant : ‘remi’, Mot de passe : ‘azerty’
```

- Code source du projet :
  https://github.com/remipassmoilesel/djoe

Ce stage a déjà fait l’objet d’un rapport préliminaire où ont été détaillés les technologies étudiées, l’organisation du projet ainsi que les premières étapes. Dans ce deuxième rapport l’attention est portée sur les étapes finales du projet et sur le détail des travaux entrepris.

Logiciel libre

L'ensemble des travaux réalisés et détaillés dans ce rapport a été réalisé à l'aide de projets libres et à sources ouvertes. Systèmes d'exploitations, langages de script et de programmation, logiciels serveur, infographies, bureautique, navigateurs, environnements de développement intégrés, tous les composants du projet et les outils utilisés sont librement téléchargeables et redistribuables.

https://www.gnu.org/philosophy/free-sw.fr.html

2. Contexte du stage

Présentation de l'entreprise Silverpeas

Silverpeas est une entreprise qui emploie 12 personnes basée à Europole. Elle conçoit un portail collaboratif sous licence libre (GPLv3) et propose des services tels que : installation et mise en œuvre, formation à l'utilisation et à l'installation, développement, adaptations graphiques, support. L'entreprise à été fondée en 2000 et a été une des premières à proposer un portail collaboratif en France.

Silverpeas a fait le choix d'utiliser et de promouvoir des technologies libres. L'entreprise propose son portail en téléchargement libre sur Github et sur le site communautaire www.silverpeas.org. Un installeur ainsi qu'une image Docker permettent à des utilisateurs non experts d'essayer la plateforme, ce qui assure une bonne visibilité à l'entreprise et à son produit.

Présentation du portail Silverpeas

Le but du portail collaboratif Silverpeas est d'améliorer les performances des organisations (associations, entreprises, services publics , ...) en optimisant la collaboration et la gestion des connaissances. Pour ce faire, elle fournit clé en main une trentaine d'applications environ:

- indexation et rassemblement d'informations sur les acteurs d'une organisation,
- gestion électronique de documents : édition en ligne non simultanée, versionnage, partage en ligne, indexation ...
- gestion de projet : création d'espaces de travail dédiés, édition de diagrammes de Gantt, ...
- organisation en interne : almanach, quiz, workflow, calendrier de partage de salles et de véhicules, ...
- concertation : enquêtes, sondages, forums, commentaires ...
- communications externes : création de sites Web, …

L'entreprise Silverpeas a fait le choix assumé d'utiliser et de proposer du logiciel libre. Cette stratégie permet aux utilisateurs potentiels de pouvoir découvrir la plateforme et son fonctionnement en effectuant une installation de découverte à l'aide d'un installeur ou d'une image Docker.

L'entreprise propose d'héberger sa plateforme en SaaS (logiciel en tant que service) ou de l'installer sur les serveurs de ses clients. En effet, bien qu'une installation de découverte soit accessible à un débutant, une installation de qualité production requiert des connaissances techniques avancées et une bonne connaissance du portail Silverpeas.

Public du portail

La plateforme Silverpeas s'adresse avant tout aux organisations de taille moyenne à grande. Le client ayant le plus d'utilisateurs emploie 30 000 personnes. Les domaines d'utilisation sont variés : industrie, santé, recherche technologique ou agricole, conseils généraux, transports publics, etc...

Étant donné que le projet est distribué sous licence libre et accessible à tous, et qu'aucune statistiques ne sont collectées, il est difficile de définir un public exact tant au niveau de l'expérience utilisateur attendue que des matériels et logiciels utilisés. C'est pourquoi l'ergonomie est prise en charge par un ergonome spécialisé et prend en compte les retours des utilisateurs.

Conception modulaire

Le portail est conçu de manière modulaire, avec un module central chargé de la gestion des ressources et de la liaison entre composants, puis plusieurs composants périphériques constituant les applications et services métier de Silverpeas.

Le portail est avant tout destiné à des postes de bureautique et n'a pas d'interface adaptative pour périphériques mobiles. Cependant, une application mobile est disponible.

Silverpeas utilise les technologies Java Édition Entreprise et HTML5. Le portail fonctionne sur un serveur d'application WildFly (la version open source de JBoss EAP).

Intégration système

Une attention particulière à été portée sur le respect des normes et standards. De ce fait, le portail s'intègre sur de nombreuses plate-formes et peut profiter des infrastructures logicielles éventuellement présentes : annuaires LDAP, bases de données métier, …

3. Objectifs du stage

L'objectif final du stage est le remplacement de la messagerie instantanée du portail Silverpeas, par la production d'une messagerie instantanée texte un pour un.

Cependant, au fur et à mesure des avancées, les objectifs fixés ont évolué :

- Messagerie texte un pour un
- Messagerie texte un pour plusieurs
- Édition collaborative de documents en temps réel
- Appels vidéo
- Vidéoconférences
- Partage d’écran
- API de manipulation de la messagerie, pour une meilleure intégration

De plus, ces contraintes précisent les objectifs :

- L'utilisation de technologies et de protocoles standardisés et ouverts
- Une compatibilité maximale avec les navigateurs Web courants
- L'utilisation de moyens de communication sécurisés
- L'utilisation de technologies compatibles avec la licence GPLv3 du portail Silverpeas
- L'utilisation de notifications claires (arrivée de message, ...)

Rappel sur l’organisation du projet

Le projet est mené en autonomie par le stagiaire sous la direction de l’architecte développeur de l’entreprise et du responsable de produit.

Publications régulières

Tous les travaux effectués sont régulièrement publiés sur un serveur de démonstration pour essai. Une feuille de route est tenue à jour à l’adresse https://im.silverpeas.net/etherpad/p/feuille-de-route

Réunions hebdomadaires

Une réunion hebdomadaire est organisée pour donner l’orientation du projet et prendre les décisions.

Logiciel libre

Le code source est publié sous licence GPLv3, géré avec l’outil Git et publié sur la plateforme Github.

https://github.com/remipassmoilesel/djoe

Le code source de l’intégration est disponible sur un des dépôts de l’entreprise Silverpeas :

https://github.com/SilverTeamWork/Silverpeas-Core/tree/chat-integration

Public visé

L'entreprise Silverpeas ne recueille pas de statistiques sur ses utilisateurs. De plus, étant donné que la plateforme est distribuée sous licence GPLv3 il est difficile de déterminer quels sont les utilisateurs cibles, et quels matériels ou logiciels ils utilisent.

Par conséquent, la messagerie devra être compatible avec un maximum de navigateurs différents et à destination d'un public varié.

Existant

L'existant sera peu détaillé car très rudimentaire. La messagerie actuelle du portail Silverpeas permet en théorie l'envoi de messages d'une personne vers une autre et la notification d'utilisateurs.

Cette messagerie instantanée à été développée ex-nihilo, sans utilisation de framework ou de protocoles existants. Elle a été développée avec les technologie d'alors c'est à dire JSP, HTML et Javascript.

En pratique la messagerie est devenue inopérante, étant donné que la messagerie utilise des fenêtres « pop-up » et que ces dernières sont systématiquement bloquées par les navigateurs modernes. De plus l'intégration de cette messagerie est peu poussée et son accès non évident.

Veille technologique et concurrentielle

Une veille technologique a été assurée pendant 3 semaines afin de repérer les projets prometteurs et les technologies actuelles (cette phase est plus détaillée dans le premier rapport). En parallèle, une veille concurrentielle a été menée auprès de professionnels de la collaboration en entreprise mais aussi auprès d'autres systèmes proposant des communications en temps réel.

En tout 12 systèmes ont été étudiés en détail, et de nombreux autres ont été survolés : Google +, Hubl'in, Jalios, Citadel, Big Blue Button … Le résultat de l'étude a fait l'objet d'un document publié sur le portail Silverpeas. Cette veille a montré tout d'abord un manque d'intérêt de la part des concurrents directs de Silverpeas pour les communications en temps réel. Ceci peut s'expliquer déjà par la nature des plate-formes proposées (orientées sur la gestion des connaissances dans le long terme) mais aussi par la complexité de mise en œuvre de communications en temps réel fiables.

De plus, cette veille technologique à montré clairement que l'utilisation de plugins tels que Flash ou Java est encore généralisée bien que fortement dépréciée. Malgré l'émergence de nouvelles technologies cohérentes et largement compatibles ces pratiques persistent.

4. Rappel sur les contraintes de communications en temps réel pour le Web

Les communications instantanées pour le Web sur un réseau public posent plusieurs problèmes. Avant de détailler le déroulement du projet il est nécessaire de prendre connaissance des contraintes inhérentes à la mise en place d'une messagerie instantanée multimédia.

Ces problèmes sont abordés ci-dessous dans le contexte du projet de stage, c'est à dire dans le cadre de l'utilisation d'un navigateur Web et de transports utilisés pour le Web.

Signalement

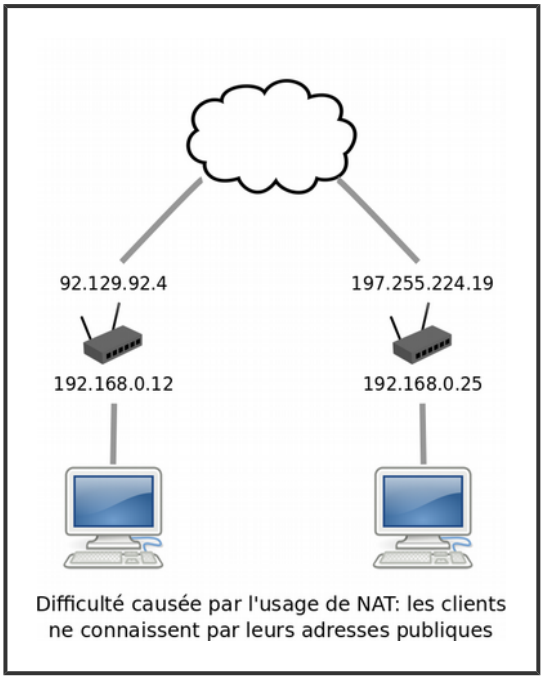
Sur un réseau où plusieurs clients souhaitent communiquer entre eux, le signalement permet aux clients de se découvrir et d'échanger des informations sur leurs capacités multimédia. Ces opérations peuvent être

intégrées dans certains protocoles et certains logiciels serveurs.

Diversités des topologies réseau

Ce point est particulièrement contraignant puisque omniprésent en entreprise et tout à fait imprévisible :

- Premièrement l'utilisation de NAT conduit un client à ne pas connaître son adresse IP publique, ce qui peut empêcher la communication entre clients. En effet si un client transmet une adresse IP privée, elle sera inutilisable sur un autre réseau
- Deuxièmement l'utilisation de pare-feux modifie ou empêche le transit d'informations par certains ports ou certains modes de transport



Après somme de toutes les informations analysées, un seuil de conditions restrictives a été fixé en dessous duquel la messagerie texte ne sera pas opérante. A partir de ce seuil sont garantis sur les réseaux :

- Le trafic HTTPS sur le port 443

Dans la majorité des situations seront disponibles également pour les échanges multimédia :

- Le trafic UDP sur le port 443
- Le trafic en TCP et UDP en paire à paire entre les clients

Dans le cas de technologies abordées ci-dessous, des logiciels serveurs spécifiques sont disponibles pour répondre à ces problématiques.

Diversité des systèmes clients

La diversité des systèmes d'exploitation et des navigateurs Web utilisés par les clients d'une application de messagerie cause de grandes disparités dans les capacités des clients et leurs compatibilités.

Les problèmes de compatibilité concernent surtout le transit de médias, le transit de texte étant plus facilement gérable. Jusqu'à très récemment pour résoudre ce problème, deux stratégies étaient employées :

- Un développement spécifique pour chaque plateforme (Linux, Mac, Windows, ...)
- L'utilisation de plugins permettant d'harmoniser les capacités des différents clients et d'ajouter un niveau d'abstraction au problème (Flash, Java, ...)

La deuxième solution n'est plus d'actualité bien qu'encore employée, puisque les plugins utilisés sont désormais activement dépréciés par les principaux navigateurs (Java et Flash).

Désormais avec les évolutions récentes des navigateurs modernes, il est désormais possible de développer des applications multimédia en faisant abstraction des différences de système d'exploitation.

5. Protocoles utilisés

XMPP

XMPP est un ensemble de protocoles standardisés dédiés aux communications en temps réel. Dérivé du XML, le XMPP est structuré et hautement extensible. Destiné premièrement à la messagerie textuelle instantanée, il a peu à peu été étendu pour supporter des fonctionnalités complexes telles que la gestion des droits d'utilisateurs ou le transfert de données multimédia. Le XMPP est particulièrement adapté aux communications à grande échelle sur des réseaux publiques décentralisés.

Identification des acteurs

XMPP prend en compte plusieurs types d'acteurs : utilisateurs, serveurs et composants (utilisateurs non-humains) Chaque acteur est identifié par une chaîne de caractères appelée Jabber ID (JID) sous la forme :

```
identifiant_utilisateur@service-xmpp/ressource

/* Où 'ressource' représente une chaîne de caractères unique par client connecté, un
   utilisateur pouvant utiliser plusieurs clients simultanément. */
```

Un identifiant Jabber est dit 'full' (fulljid) lorsque qu'il contient la dernière partie de l'identifiant, la ressource. La ressource est une chaîne de caractères qui peut avoir plusieurs signification (parfois le nom du client, parfois un lieu : /jsxc, /jitsi, /homee, /desktop…) Dans la plupart des communications la ressource désigne de manière unique un client à partir d'une chaîne aléatoire de caractère générée par le serveur. Cette information est essentielle pour les appels multimédia.

Les bases du protocole

L'ensemble des standards XMPP reposent sur trois types de messages. Les messages sont appelés des « stances ».

<presence>

La stance de présence est utilisée pour envoyer régulièrement des informations sur le statut d'un client. Les statuts peuvent être textuels, à destination d'utilisateurs humains, ou codifiés à destination de machines.

Exemple de présence :

```
<presence>
      <show>away</show>
      <status>at the ball</status>
</presence>
```

<message>

La stance message est le transport privilégié d'informations à destination d'utilisateurs. Il peut être de type :

- 'chat' : communication entre deux utilisateurs
- 'groupchat' : communication entre utilisateurs dans un groupe de discussion
- ou de types plus spécifiques pour les communications de service

Le message peut comporter un ou plusieurs éléments « body » qui sont les seuls devant être affichés, sous certaines conditions. De cette manière, la stance message est facilement extensible pour pouvoir supporter par exemple le transport de métadonnées (favoris, mots clefs, ...).

La stance de message peut également transporter des média. En effet dans le cas de l'utilisation de certains modes de transport, un flux binaire peut être converti en base 64 et intégré dans une stance de message.

Exemple de message :

```
<message from='bingley@netherfield.lit/drawing_room'
       to='darcy@pemberley.lit'
       type='chat'>
       <body>Come, Darcy, I must have you dance.</body>
       <thread>4fd61b376fbc4950b9433f031a5595ab</thread>
</message>
```

<iq>

IQ pour « Information / Query » est une stance destinée aux requêtes de service XMPP. Un exemple parlant de l'utilisation de stances IQ est la découverte de services (XEP 0030). Un client XMPP connecté à un réseau peut découvrir les services disponibles offerts par le serveur pour adapter son comportement.

Exemple de requête IQ :

```
< ! – Requête de découverte de services -->
<iq type='get'
    from='romeo@montague.net/orchard'
    to='plays.shakespeare.lit'
    id='info1'>
  <query xmlns='http://jabber.org/protocol/disco#info'/>
</iq>

< !– Réponse -->
<iq type='result'
    from='plays.shakespeare.lit'
    to='romeo@montague.net/orchard'
    id='info1'>
  <query xmlns='http://jabber.org/protocol/disco#info'>

    ...

    < !-- Services offerts -->

    <feature var='http://jabber.org/protocol/disco#info'/>
    <feature var='http://jabber.org/protocol/disco#items'/>
    <feature var='http://jabber.org/protocol/muc'/>

    ...

  </query>
</iq>
```

X-Forms

Pour compléter ces stances et pour conserver une architecture globale largement compatible, les X-Forms sont utilisés dans les négociations. Un X-Forms est un formulaire XML permettant de spécifier quelles informations et quels formats d'informations sont attendus en réponse à une communication. L'utilisation de X-Forms permet une bonne structure de communication indépendante des différentes implémentations possibles de composants.

Exemple de recherche d'utilisateurs en utilisant X-Forms :

```
< !-- Le client qui souhaite effectuer une recherche envoi une première requête, et le
       serveur lui retourne un formulaire X-Forms. Ce formulaire est ensuite rempli puis
       renvoyé -->

<iq type='set'
    to='search.im.silverpeas.net'
    xmlns='jabber:client'
    id='de6a30a5-d3f3-4f28-bddd-33725ab86666:sendIQ'>

       <query xmlns='jabber:iq:search'>

              <x xmlns='jabber:x:data' type='submit'>
              <field type='hidden' var='FORM_TYPE'>
                     <value>jabber:iq:search</value>
              </field>

              < !-- Critére : le champs doit commencer par 'An' -->

              <field var='search' type='text-single'>
                     <value>An*</value>
              </field>

              < !-- Les champs 'identifiant' et 'nom' seront utilisé (value=1) →

              <field var='Username' type='boolean'>
                     <value>1</value>
              </field>
              <field var='Name' type='boolean'>
                     <value>1</value>
              </field>
              </x>

       </query>
</iq>
```

Extensions du protocole: les XEP

Les fonctionnalités du protocole XMPP sont basiques mais sont étendues par de nombreuses XEP : XMPP Extension Protocol. Une XEP est une spécification d'extension du protocole XMPP.

XMPP est un standard ouvert. Tout le monde peut proposer sa propre extension du protocole pour publication à la XMPP Standard Foundation à condition qu'elle respecte la XEP des XEP, la XEP 0001 « XMPP Extension Protocols ». En fonction de l'avancée d'une XEP et de sa compatibilité avec les autres XEP son état évolu de «expérimentale» à « active » ou à «obsolète».

Transports possibles

Pour acheminer des stances XMPP plusieurs types de transports sont disponibles :

- en « natif », c'est à dire en TCP à travers le port 5222. Méthode originelle non adaptée au Web car non supportée par les navigateurs

- **par WebSocket :** le choix moderne de plus en plus apprécié. Cette solution est efficace mais encore indisponible pour certains navigateurs, et parfois bloquée par des pare-feux
- **par HTTP avec BOSH** (Bidirectional-streams Over Synchronous HTTP). Cette solution bien que plus ancienne et consommant plus de ressources est compatible avec tous les navigateurs Web, et passe tous les pare-feux permettant le trafic HTTP/HTTPS

De plus, le XMPP permet le transport de données multimédia. Le XMPP étant un format textuel il peut poser problème pour le transport de données binaires. Dans ce cas, deux types de transports sont possibles :

- **« out-of-band » :** le plus efficace. Le transport est négocié entre deux client en XMPP mais le flux média est transmis par un moyen alternatif, parfois en paire à paire, parfois par un serveur relais
- **« in-band » :** le flux de données est converti en base 64 puis segmenté et intégré dans le corps de stances « message ». Cette méthode est gourmande en ressources mais présente l'intérêt de pouvoir passer quasiment tous les pare-feux si le serveur est correctement configuré

Infrastructure nécessaire

Un serveur XMPP : pour l'échange de données textuelle

Application au projet

XMPP est le meilleur choix dans le cadre de ce projet pour la messagerie textuelle mais aussi en tant que composant de la messagerie multimédia. Dans le cadre de la messagerie textuelle XMPP offre une excellente structure normée et extensible. De plus, étant donné que ces standards sont ouverts de nombreux logiciels serveurs de qualité professionnelle sont disponibles et implémentent des fonctionnalités complexes de la messagerie instantanée (messagerie de groupe, gestion des droits d'utilisateurs, ...)

En complément, dans le cadre de la messagerie multimédia, XMPP fournit une méthode de signalement efficace et permettra la découverte de paire et la négociation de flux multimédia entre les clients.

WebRTC

WebRTC est une interface de programmation applicative permettant de manipuler des flux de médias entre navigateurs Web en paire à paire. WebRTC se présente sous la forme d'un ensemble d'outils accessibles en Javascript. L'objectif de cette technologie est de simplifier l'utilisation de flux multimédia dans les navigateurs, sans utilisation de plugins externes.

WebRTC permet d'échanger des flux multimédias variés (texte, flux binaires arbitraires et flux vidéos) mais est particulièrement optimisé pour la manipulation de vidéos. WebRTC ne spécifie pas les méthodes de transport ou de signalement dans un but de non redondance : la priorité est laissée aux technologies existantes. Cependant, dans la pratique le transport UDP est privilégié bien que d'autres modes soient disponibles.

Objectif de simplification mais infrastructure lourde

Bien que WebRTC soit censé rendre autonome les navigateurs Web pour le transit de médias en paire à paire, plusieurs difficultés font que l'infrastructure serveur nécessaire à l'utilisation de cette API est lourde.

En effet, étant donné la diversité des topologies réseau, et notamment l'utilisation courante de NAT et de pare-feux, plusieurs logiciels serveurs sont nécessaires.

On peut distinguer deux embûches majeures dans les communications WebRTC :

- **Un client est derrière un NAT et ne connaît pas son adresse IP publique**

  Dans ce cas lorsque que le client A négocie une connexion avec le client B, le client A envoi son adresse privée qui est inutilisable pour le client B. Pour pallier à ce problème il est possible de mettre à disposition des clients un serveur **STUN** (Simple Traversal of UDP through NATs). Ainsi, en effectuant une requête vers un de ces serveurs un client pourra découvrir son adresse publique.

  Un serveur STUN est peu consommateur de ressources et son utilisation permet à 86 % des connexions initiées de réussir (source : www.html5rocks.com)

- **Un client est derrière un pare-feu et ses communications sont limitées**

  Dans cette catégorie de cas variés, un client peut ne pas avoir accès à une certaine plage de ports, ou le trafic peut être limité au transport TCP. Pour pallier à ce problème, il est possible de mettre à disposition des utilisateurs un serveur **TURN** (Traversal Using Relays around NAT)**.** Le serveur TURN, en plus de proposer les services d'un serveur STUN, permet de relayer des médias en transit entre deux destinataires. Un serveur TURN peut adapter les modes de transports en fonction de son environnement.

  Le travail d'un serveur TURN peut être lourd si beaucoup de clients l'utilisent simultanément mais l'utilisation de ce serveur permet le transit de médias dans beaucoup de cas difficiles et avec de bonnes performances.

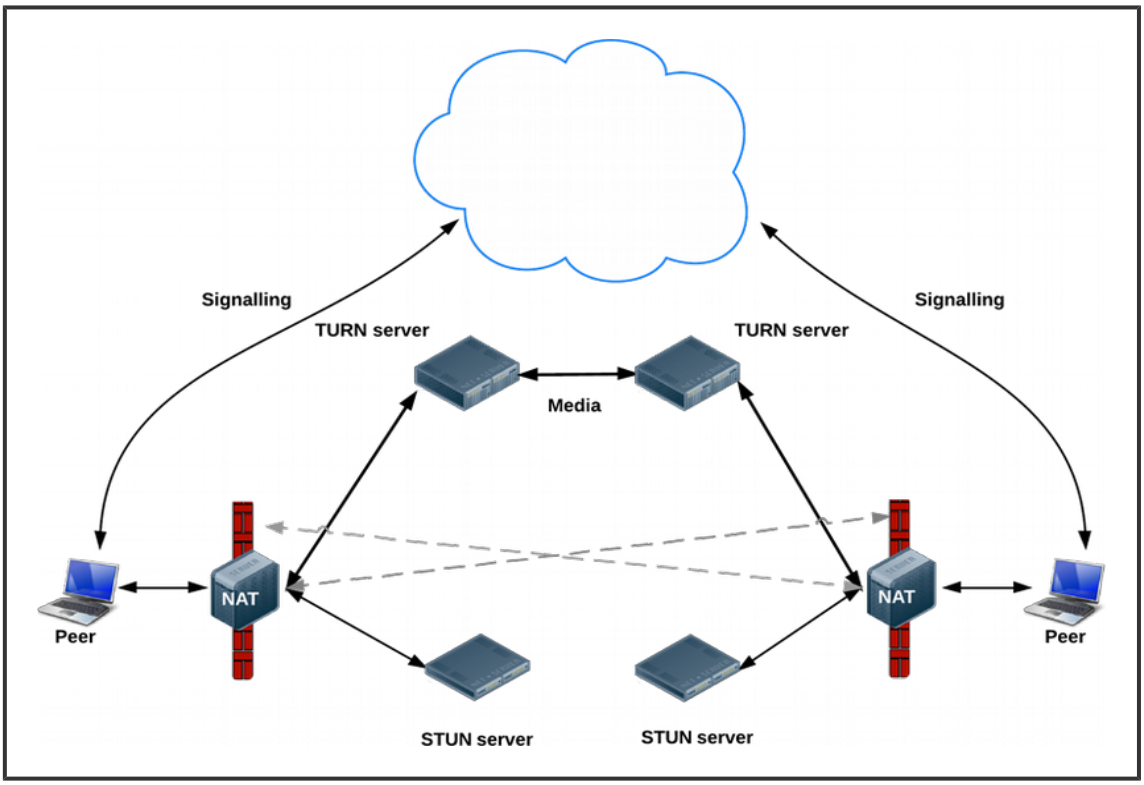
Procédure ICE

ICE (Interactive Connectivity Establisment) est une procédure de détermination automatique du meilleur moyen pour deux clients de se joindre. Plusieurs solutions sont étudiées successivement pour déterminer la meilleure, de la moins contraignante pour les serveurs distants à la plus contraignante.

Globalement la procédure ICE se déroule comme suit :

- Les clients essaient de se connecter en paire à paire avec les adresses à leur disposition
- En cas d'échec chaque client effectue une requête STUN pour obtenir son adresse publique et retente une connexion
- En cas d'échec le serveur TURN relaie le flux entre les clients sur une plage définie de ports (par défaut [49152-65535])

Schématisation d'une installation STUN / TURN



Source : http://www.html5rocks.com/en/tutorials/webrtc/infrastructure/

Infrastructure nécessaire

Serveur de signalement : pour la découverte de paires

Serveur TURN : pour permettre aux clients de découvrir leurs adresses publiques et éventuellement pour faire relais entre deux clients situés dans des environnements contraignants

Serveur HTTP utilisant TLS (HTTPS): indispensable pour que les navigateurs autorisent l'accès aux périphériques de capture multimédia

Serveur MCU (Multi Control Unit): optionnel, permet la transformation des flux vidéos à la volée. Les transformations possibles sont nombreuses mais la plupart sont destinées à adapter la vidéo aux périphériques cibles (ex : mobiles) ou à l'ajout d'informations (ex : réalité augmentée)

Application au projet

WebRTC est le meilleur choix disponible pour le transport de données multimédia. Cependant, l'infrastructure logicielle nécessaire est lourde et WebRTC étant une norme récente, elle est susceptible d'évoluer rapidement. Il peut être risqué d'appuyer tout un projet de messagerie sur cette norme dans son état actuel.

Externalisation des services multimédias

A la fin de la période de veille une question s'est posée naturellement : Quel sera le meilleur choix en terme d'infrastructure serveur ? Proposer des services multimédia ou les externaliser ?

L'infrastructure nécessaire aux services multimédias est réputée complexe à mettre en œuvre et très gourmande en ressources. Pour simplifier l'utilisation de ces services plusieurs sociétés proposent des API en SaaS (logiciel en tant que service) permettant de se libérer de ces contraintes à des prix compétitifs.

Cependant une externalisation totale des services de messagerie instantanée est inenvisageable car

incompatible avec la philosophie du portail Silverpeas qui est de fournir une solution clef en main complète. De plus certains clients ayant des politiques strictes de sécurité il pourrait être délicat d'imposer l'utilisation de serveurs externes à une entreprise.

Dans le but d'optimiser la qualité du service de messagerie multimédia, sans passer à coté de l'occasion de proposer éventuellement un service valorisable, une démarche à été convenue :

1. Mettre en place une messagerie texte structurée qui sera disponible dès l'offre la plus basique sur le portail Silverpeas
2. Puis dans un premier temps tenter de mettre en place l'infrastructure nécessaire aux communications multimédias en temps réel. Des tests seront ensuite effectués pour observer la quantité de ressources consommées
3. Et dans un deuxième temps si la charge se révèle conséquente, découpler la gestion multimédia du client pour permettre l'alternance entre l'utilisation de services internes et externes. Ainsi l'offre de messagerie se divisera en deux :
    - Une messagerie texte et multimédia basique comprise dans l'offre de portail
    - L'utilisation de services externes en options pour une utilisation multimédia intensive

6. Détail du projet

JSXC : un point de départ structuré

JSXC pour « Javascript XMPP Client » est un client XMPP disponible sous licence MIT créé en 2012 par un étudiant Allemand de l'université de Constance. Conçu comme une application indépendante, ce client est tant dans sa structure que dans son interface destiné à être intégrable dans n'importe quelle page Web. Le projet JSXC s'est notamment fait connaître pour avoir publié des plugins populaires pour OwnCloud, RoundCube, ainsi que d'autres plate-formes.

Ce client n'utilise pas les dernières technologies Web mais malgré ses défauts le projet est très bien structuré, clair, et correctement documenté.

Le projet original propose comme fonctionnalités :

- messagerie texte un pour un
- messagerie texte un pour plusieurs
- utilisation de la messagerie sur plusieurs onglets simultanément
- notifications sonores et de bureau
- cryptage de conversation de client à client « Off The Record », permettant de protéger l'information même si elle passe à travers des serveurs publics ou non sécurisés
- appels vidéo un pour un
- transfert de fichier

Une allocution de Klaus Herberth, l'auteur du client JSXC, à la conférence OwnCloud 2015 est disponible en vidéo: https://www.youtube.com/watch?v=tFY7kTF3eDs

De JSXC à Djoe, quelle évolutions ?

Tous les travaux de ce stage ont été rassemblé dans un nouveau projet, baptisé Djoe. La volonté de créer un projet distinct de JSXC est motivée par les différences d'objectifs des deux projets :

- JSXC est un client XMPP, destiné au réseau public et qui aspire a être interopérable avec n'importe quel autre client XMPP
- Alors que ce projet est un système complet, infrastructure logicielle serveur comprise, destiné à devenir un réseau privé où tous les clients sont identiques

Ce qui a été développé et ajouté à la version modifiée de JSXC :

- Recherche d'utilisateurs partageant le même domaine XMPP
- Discussion avec des utilisateurs non contact
- Indicateur de frappe indiquant qui est en train d'écrire dans une conversation
- Création de conversation en un clic avec invitation de participants
- Signalisation de présence plus précise avec envoi automatique de présence lorsque l'utilisateur utilise l'interface graphique
- Un module de création et de lancement de guides interactifs
- Édition collaborative grâce au système Etherpad
- Vidéoconférence jusqu'à 6 participants
- Partage d'écran sur Chromium/Chrome
- Module API permettant de manipuler le client JSXC

Ce qui a été abandonné et désactivé dans la version modifié de JSXC :

- Le cryptage « Off TheRecord » : permet de chiffrer des communications textuelles de client à client, même sur une connexion HTTP. Ce module est source d'erreur et a été jugé inutile étant donné la nature du réseau cible (sans serveurs publics et utilisant HTTPS)
- L'utilisation simultanée du client sur plusieurs onglets : source de confusion pour l'utilisateur, une distinction est faite entre le premier onglet ouvert et les autres. Le premier onglet ouvert possède des fonctionnalités que les autres onglets n'ont pas, sans indications claires de ces différences
- Le transfert de fichier en paire à paire, qui n'est pas prioritaire
- Le module WebRTC original, remplacé par un nouveau module

Deux versions du client en essai pour comparaison

Pour se faire une idée des évolutions du client JSXC modifié, une version du client original est disponible en essai : https://im.silverpeas.net/jsxc.original/example/

Informations de connexion :

```
XMPP domain : im.silverpeas.net
BOSH url : https://im.silverpeas.net/http-bind/
AJAX Login : Identifiant : ‘remi’, Mot de passe : ‘azerty’
```

Bien entendu, comme indiqué au début de ce document, la version modifiée du client JSXC est disponible sur la plateforme de démonstration : https://im.silverpeas.net

Chronologie succincte du projet

Cette chronologie succincte résume le calendrier de l’évolution du projet :

Mai 2016

- 3 semaines de veille technologique et d’essais de plusieurs plate-formes
- Installation et mise en ligne de deux projets pour essai : JSXC et Kaiwa. JSXC est retenu comme meilleur candidat à l’intégration
- Création et mise en place d’outils de diagnostic Web et XMPP
- Adaptation de l’environnement de développement de JSXC
- Mise en place d’une première infrastructure serveur sécurisée

Juin

- Adaptation de l’environnement de développement
- Premières manipulations de XMPP et premières modifications du client JSXC
- Premier changement d’interface du client JSXC
- Première démonstration de la vidéoconférence devant un client potentiel de l’entreprise Silverpeas
- Création des « conversations » et envoi d’invitations
- Amélioration de l’infrastructure serveur, ajout du proxy Apache
- Première intégration d’Etherpad pour essai

Juillet

- Première intégration de la messagerie dans le portail Silverpeas 6
- Ajout du module d’interface de programmation applicative
- Création d’un module de statistiques pour suivi de l’utilisation des fonctionnalités
- Remplacement du module WebRTC orignal du projet JSXC
- Premiers appels vidéo multiples
- Création d’une première procédure de vidéoconférence, trop peu fiable
- Deuxième procédure de vidéoconférence plus travaillée avec suivi d’état des clients
- Adaptation d’une extension de capture d’écran de Chrome

Août

- Poursuite de l’intégration dans le portail Silverpeas
- Ajout d’un utilitaire de journalisation distante des erreurs
- Création du module de tests unitaires pour la vidéoconférence
- Partage d’écran disponible sur Chrome et Chromium
- Ajout du système d’invitations Etherpad
- Deuxième changement d’interface
- Ajout d’un filtre de message pour mise en valeurs de ressources (liens, vidéos, …)

Création et mise en place d'outils de diagnostic

Dans le but de mieux diagnostiquer les problèmes lors des installations de serveurs et du développement plusieurs outils ont été mis en place. Tout ces outils sont directement accessibles à partir de la plateforme de démonstration, en bas de page section « Informations de développement » : https://im.silverpeas.net/

- **Un observateur de trafic XMPP :** Un plugin JQuery développé pour l'occasion permettant d'afficher le trafic en cours d'une application XMPP. Utilisé pour apprendre le fonctionnement de XMPP et repérer d'éventuels problèmes. Disponible à l’adresse: https://im.silverpeas.net/ (en bas de page). Code source : https://github.com/remipassmoilesel/xmpp-inspector
- **Un testeur de connexion BOSH :** Une application Web simple permettant de diagnostiquer une connexion XMPP à travers HTTP. Travail dérivé du client JSXC. Disponible à l’adresse: https://im.silverpeas.net/bosh-test/
- **Un testeur de connexion ICE :** Une application Web simple permettant de diagnostiquer une connexion STUN / TURN (S). Travail dérivé de https://webrtc.github.io. Disponible à l’adresse: https://im.silverpeas.net/ice-test/
- **Une console d’événements JQuery :** Un plugin JQuery développé pour l'occasion permettant l'observation des événements JQuery émis sur une page. Disponible à l’adresse: https://im.silverpeas.net/ (en bas de page) Code source : https://github.com/remipassmoilesel/jquery.eventconsole
- **Une console XMPP :** Une application Web permettant d'envoyer et de recevoir des stances XMPP. Cette console permet notamment de vérifier la syntaxe des requêtes envoyées et de visualiser les échanges. Ce travail est dérivé du livre « Professional XMPP Programming with JavaScript and jQuery ». Disponible à l’adresse : https://im.silverpeas.net/xmpp-console/
- **XMPP-Disco :** Une application Web simple qui utilise la XEP 0030 Service Discovery permettant la découverte de capacités d'un serveur XMPP. Travail dérivé du livre « Professional XMPP Programming with JavaScript and jQuery ». Disponible à l’adresse: https://im.silverpeas.net/xmpp-disco/

Mise en place de l’infrastructure logicielle serveur

La messagerie instantanée texte et multimédia nécessite l’usage de plusieurs logiciels serveurs. Dès le début du projet il est apparu que le développement devrait se dérouler au plus près des conditions de production, afin de détecter le plus tôt possible les problèmes.

Configuration matérielle

Le serveur de démonstration actuel est une machine virtuelle contrôlée par VMWare disposant de 8 cœurs à 2,5 Ghz, 8 Go de mémoire RAM, et 13 Go d'espace disque. Cette configuration est sous utilisée par le système de messagerie en lui même et les logiciels serveurs utilisés, mais la plateforme Silverpeas 6 et son serveur Wildfly requièrent une bonne configuration serveur.

Les contraintes

- **Utilisation de TLS généralisée** : L'utilisation de communications sécurisées est indispensable pour que le client XMPP puisse être utilisé sans problème. En effet les navigateurs empêchent le chargement de ressources non sécurisées dans certains cas, et empêchent l'accès aux périphériques d'acquisition multimédia.
- **Un domaine distinct et deux interfaces réseaux supplémentaires pour TURN** : En effet, pour que le serveur STUN/TURN puisse découvrir au mieux le type de réseau utilisé par ses clients, les clients doivent contacter deux adresses IP distinctes sur les ports 80 et 443.

Fichiers de configuration

Tous les fichiers de configuration sont régulièrement récupérés grâce à Rsync et versionnés grâce à Git. Ce qui permet un suivi détaillé des changements ainsi qu'une installation facilitée.

Système de pare-feu

Un script d'établissement de règles IPTables a été réalisé pour protéger le serveur de démonstration. Ce script, adapté à partir d'un script existant réalisé par l'administrateur système de l'entreprise Silverpeas, permet notamment le trafic :

- UDP et TCP sur les ports 80 et 443
- UDP et TCP sur une plage de ports allant du port 49152 au port 65535

Extrait du script de pare-feu :

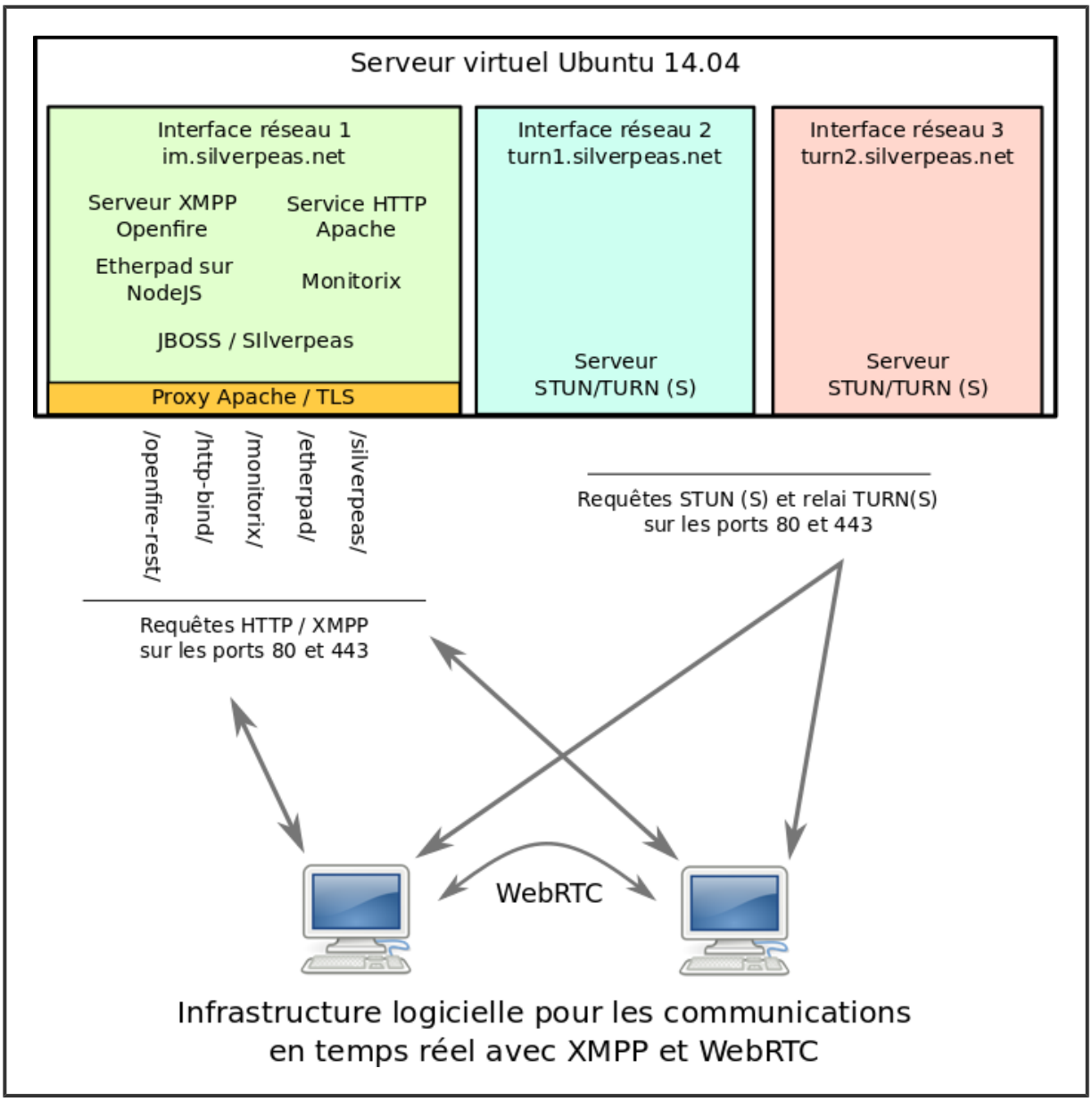
```
# HTTP + HTTPS + STUN + TURN
iptables  -A INPUT -p tcp --dport 80 -j ACCEPT
iptables  -A INPUT -p tcp --dport 443 -j ACCEPT
iptables  -A INPUT -p udp --dport 80 -j ACCEPT
iptables  -A INPUT -p udp --dport 443 -j ACCEPT

# STUN / TURN
iptables  -A OUTPUT -p udp --dport 49152:65535 -j ACCEPT
iptables  -A INPUT  -p udp --dport 49152:65535 -j ACCEPT
iptables  -A OUTPUT -p tcp --dport 49152:65535 -j ACCEPT
iptables  -A INPUT -p tcp --dport 49152:65535 -j ACCEPT
```

Installation facilitée sous GNU/Linux

L'installation de cette infrastructure a été longue et a demandé beaucoup de configuration mais cette mise en place a heureusement été grandement simplifiée par l'utilisation d'Ubuntu 14.04 et de son gestionnaire de paquets (APT)

Schéma de l’infrastructure logicielle serveur



Détail de l'infrastructure

- **Un serveur Apache :** Pour servir le client XMPP et le site de démonstration de la messagerie, mais surtout en tant que proxy TLS pour toutes les applications. Grâce à l'utilisation du proxy Apache les requêtes sont centralisées, et tous les serveurs peuvent écouter simultanément les ports 443 et 80 ce qui est essentiel pour permettre une accessibilité maximale vis à vis des pare-feux. De plus le proxy Apache facilite la gestion des clefs TLS, et permet de résoudre les problèmes récurrents de Cross Origin Resource Sharing.
- **Un serveur XMPP Openfire** : pour gérer les communications textuelles mais aussi comme serveur de signalement multimédia grâce à la XEP 0166 Jingle.
- **Un serveur STUN/TURN** : pour permettre aux clients de découvrir leurs adresses IP publiques et pour faire relais entre deux clients dans les cas les plus restrictifs. Le trafic TURN doit être

disponible en sécurisé pour les navigateurs compatibles, mais aussi en non sécurisé pour les navigateurs non-compatible TURNS.

- **Monitorix**: Application de surveillance de l'activité d'un système. Monitorix a été installé dans le but de surveiller la consommation de ressources du relais audio/vidéo TURN. Malheureusement ses mesures sont faussées car l'application n'est pas adaptée aux machines virtuelles (les mesures concernent le serveur entier et non la machine virtuelle).
- **Une base de donnée Postgres** : pour stocker les informations d'Openfire, du portail Silverpeas et d'Etherpad.
- **Etherpad** : pour permettre l'édition de texte collaborative.
- **Un serveur Node/Express** pour la collecte de données et de journaux d'erreur.

Adaptation de l'environnement de développement

L'environnement de développement a été adapté pour les besoins du projet. Alors que l'environnement de JSXC se limite au client lui-même, il a été décidé d'élargir le contexte du projet en accord avec ses objectifs et pour faciliter de futures installations :

- En conservant tous les fichiers du projet dans une structure cohérente pour un système GNU/Linux compatible Debian ('/etc', '/var', '/opt', ...)
- Et en conservant et en versionnant tous les fichiers de configuration modifiées

Désormais le projet est organisé comme suit :

```
.
├── etc
│   ├── ...
│   └── Fichiers de configurations des serveurs Apache, TURN, ...
├── opt
│   ├── ...
│   └── Dépendances additionnelles et documentation
└── var
    └── www
        └── djoe
            └── ... Les fichiers du client servi par Apache
```

Grâce à cette structure, lisible pour un administrateur système, il sera plus simple de mettre en place le projet sur un serveur virtuel, dans une image Docker ou pourquoi pas sous la forme d'un paquet distribuable (DEB, RPM, …). Dans le but de rapatrier les fichiers de configuration facilement, une tâche Gulp / Rsync spécifique a été conçue :

```
var gulp = require('gulp');
var shell = require('gulp-shell');

var server = "im.silverpeas.net";

gulp.task(
    'get-configuration',
    shell.task([
       'rsync -av --files-from=opt/djoe-utils/configuration.list ' + server + ':/ .'])
);
```

De même pour publier les modifications effectuées :

```
gulp.task('deploy-demo',
    shell.task([
       'rsync -avz "var/www/djoe/" ' + server + ':"/var/www/djoe/"']));
```

Adaptation de l'empaquetage du client

Dans le but de simplifier l'intégration du client, les dépendances du client JSXC ont été réduites au minimum, et l'empaquetage a été adapté.

Code d'intégration original de JSXC :

```
<link href=".../bootstrap.min.css" media="all" rel="stylesheet" type="text/css" />
<link href=".../jquery-ui.min.css" media="all" rel="stylesheet" type="text/css" />
<link href=".../jsxc.css" media="all" rel="stylesheet" type="text/css" />

<script src=".../jquery.min.js"></script>
<script src=".../jquery.ui.min.js"></script>
<script src=".../jquery.slimscroll.js"></script>
<script src=".../jquery.fullscreen.js"></script>
<script src=".../jsxc.dep.js"></script>
<script src=".../bootstrap.min.js"></script>
<script src=".../jsxc.min.js"></script>
```

Code actuel :

```
<link href=".../jquery-ui.css" media="all" rel="stylesheet" type="text/css"/>
<link href=".../jsxc.css" media="all" rel="stylesheet" type="text/css"/>

<script src=".../jquery.js"></script>
<script src=".../jquery-ui.js"></script>

<script src=".../jsxc.dep.js"></script>
<script src=".../jsxc.js"></script>
```

Il est également possible d'empaqueter optionnellement JQuery avec JSXC pour réduire encore le code d'intégration :

```
<link href=".../jsxc.css" media="all" rel="stylesheet" type="text/css"/>
<script src=".../jsxc.dep.js"></script>
<script src=".../jsxc.js"></script>
```

La structure du client sera encore optimisée, notamment en se séparant définitivement de Bootstrap et en remplaçant certaines dépendances (slimscroll, etc…) Il serait envisageable d'arriver à terme à un seul script et à une seule feuille de style. Cependant, ce n'est pas la meilleure stratégie pour favoriser le développement du client. En effet l'empaquetage de toutes les dépendances et du style est long, recommencer cet empaquetage à chaque modification serait gênant. De plus, à chaque modification du client chaque utilisateur de la messagerie serait obligé de re-télécharger l'intégralité du client, alors qu'avec des dépendances séparées seul ce qui est modifié est téléchargé à nouveau.

Suivi de l’utilisation des fonctionnalités

Un module de statistiques a été ajouté au client JSXC. Ce module recueille et transmet des « événements » textuels arbitraires a un serveur distant. Le module utilise Angular, ChartJS, NodeJS / Express et PostgreSQL.

```
jsxc.init: 2327
jsxc.etherpad.opened: 79
jsxc.help.tutorial.interface: 63
jsxc.mmstream.screenSharing.sendInvitation: 51
jsxc.mmstream.screensharing.invitationReceived: 20
jsxc.mmstream.videocall.simplecall: 11
jsxc.mmstream.videoconference.accepted: 248
jsxc.mmstream.videoconference.decline: 8
jsxc.mmstream.videoconference.invitationReceived: 355
jsxc.mmstream.videoconference.re-invitationReceived: 8
jsxc.mmstream.videoconference.sendInvitation: 503
jsxc.mmstream.videoconference.start: 166

(données issues d’une phase de développement, donc non significatives)
```

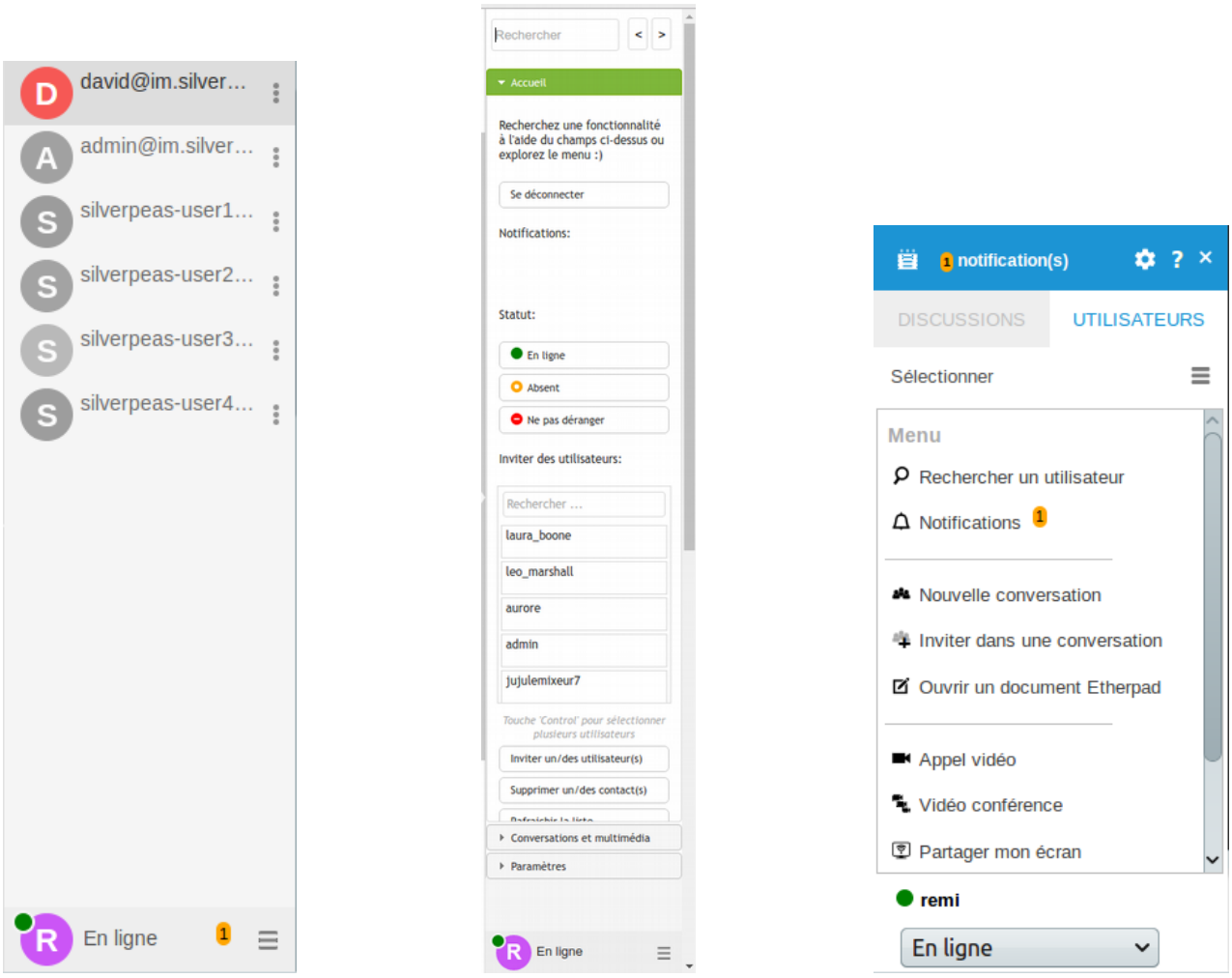
Ces événements sont associés à un numéro de session anonyme. Ils seront utilisés comme statistiques d’utilisation et permettront de prendre de bonnes décisions pour l’orientation future de la messagerie. Des graphiques sommaires permettent de se faire une idée du potentiel d’information disponible, même si ils sont à améliorer : https://im.silverpeas.net/stats/visualization/

Code source: https://github.com/remipassmoilesel/web-stats

Travail d’interface

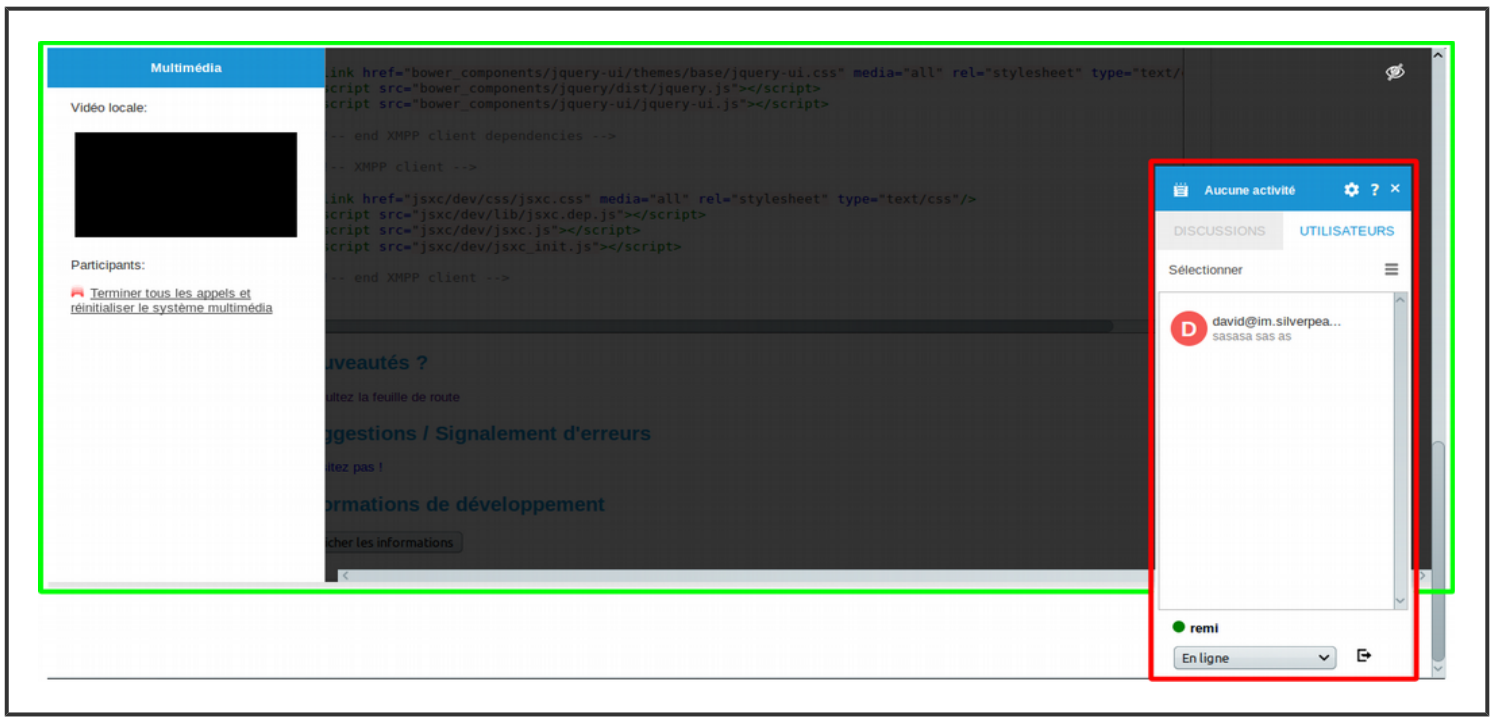
Le client JSXC original a subit deux changements d’interface successifs. Une première interface a été créé au début du développement pour tester le potentiel de modification du client JSXC. Dans un deuxième temps, après réception d’une maquette crée par l’ergonome de l’entreprise Silverpeas, une nouvelle interface a été développée et intégrée.

L’interface finale a été conçue pour être autonome et intégrable sur n’importe quelle page Web, peut importe sa structure (hors utilisation des « frameset »). Une attention particulière a été porté sur la clarté de l’interface et sur l’animation des différents éléments pour faciliter la lecture globale de l’utilisateur.



Évolutions successives de l'interface, de gauche à droite

Présentation de l'interface finale



Interface finale du client JSXC modifiée

L’interface est divisée en deux éléments principaux repliables :

- **La barre de conversation (en rouge):** élément principal toujours visible, qui permet de déclencher les principales fonctionnalités et de configurer l’application
- **Le panneau multimédia (en vert) :** permettant d’afficher des ressources sur une grande surface (vidéos, documents Etherpad, …) Le panneau est conçu pour être utilisable sur grand écran ou sur écran à proportions horizontales (ordinateurs portables, tablettes, ...)

Guides interactifs

Un module de création et de lancement de guides interactifs a été ajouté à la messagerie. Grâce au système Tether Shepherd, écrit en Javascript, il est possible de guider les utilisateurs à travers une succession d’étapes pour leur montrer le fonctionnement d’une application ou d’une fonctionnalité.

Un parcours guide est une succession d’objets ‘étape’ qui peuvent modifier un document Web, interagir avec l’interface du client JSXC modifié, et réagir à des actions de l’utilisateur sur l’interface.

Pour l’instant seul un parcours a été développé pour présenter l’interface, mais plusieurs autres sont à l’étude.

Gestion des informations de présence

Dans le but d’avoir des informations régulières sur l’état des utilisateurs de la messagerie, plusieurs aménagements sur l’envoi de messages de présence ont été effectués.

L’envoi de stances « presence » fournit des informations essentielles. Non seulement les stances décrivent l’état d’un client (connecté, déconnecté, libre, occupé, …) mais elles renseignent également l’adresse exacte du client connecté (fulljid : node@domain/resource)

Les erreurs d’informations de présence peuvent être source de confusion pour les utilisateurs : messages envoyés dans le ‘vide’, appels vidéo vers un mauvais identifiant (ce qui n’amène parfois pas de réponse), etc...

Envoi d’une présence de déconnexion

Le client JSXC original envoie une présence de déconnexion uniquement si l’utilisateur se déconnecte volontairement. Dans la version modifiée, à la fermeture d’un onglet ou du navigateur, le client essaie d’envoyer une présence de déconnexion pour prévenir les autres utilisateurs que le client n’est plus connecté.

Envoi automatique de présences

Le client JSXC modifié envoi automatiquement des messages signalant sa présence sur le réseau :

- Toutes les 15 secondes lorsque le client est connecté
- Et lorsque l’utilisateur utilise l’interface, c’est à dire lorsque la souris survole l’interface du client, dans les limites d’un nombre d’envoi maximum fixé

Ainsi, l’information disponible est toujours fraîche, même si un utilisateur se déconnecte une ou plusieurs fois par inadvertance (ce qui changera à chaque fois son identifiant, et pourra poser des problèmes).

Précision de l’information

De plus, dans l’avenir il pourrait être intéressant de préciser l’information pour l’utilisateur, par exemple en ajoutant le libellé « Actif » à un utilisateur qui utilise sa messagerie, ou absent si celui ci ne l’utilise plus.

Recherche d’utilisateurs

Pour faciliter la mise en relation d’utilisateurs sur le réseau il a été décidé de permettre la recherche et la discussion simple entre utilisateurs d’un même domaine XMPP. Dans ce but, la XEP 0055 « Jabber search » a été implémentée et ajoutée au client JSXC. De plus, le client a été modifié pour permettre la discussion entre deux utilisateurs même si ils ne sont pas entrés en contact auparavant.

La XEP 0055 fonctionne grâce aux XForms, un procédé d’échange de formulaires XML. Le client demande un formulaire de recherche, le rempli et le retourne au serveur pour obtenir d’éventuels résultats. L’emploi des XForms dans cette situation oblige le client à s’adapter au serveur, qui fixe les champs et les règles de recherche.

```
<!-- Requête de recherche d’utilisateurs dont le nom commence par ‘jul’ →

<iq type='set' from='romeo@montague.net/home' to='characters.shakespeare.lit'
      id='search2' xml:lang='en'>

      <query xmlns='jabber:iq:search'>
            <last>jul*</last>
      </query>

</iq>
```

Conversations

Dans le but de simplifier les interactions des utilisateurs entre eux, il a été décidé de faire abstraction de la notion de salon dans l’expérience utilisateur. Un salon XMPP est un espace de discussion dont les participants doivent connaître l’identifiant et qu’ils doivent rejoindre volontairement.

En utilisant les salons, la chronologie pour que les utilisateurs A, B et C discutent est la suivante :

1. A crée un salon
2. A communique l’identifiant du salon à B puis à C
3. B rejoint le salon à l’aide de son identifiant
4. Idem pour C

Pour simplifier ce système, la création des salons et la communication de leurs identifiants sont désormais gérés par le client. Ainsi pour l’exemple précédent :

1. A sélectionne dans sa liste de contacts B et C
2. A crée une ‘conversation’ (un salon dont l’identifiant est généré par le client JSXC) et attend que B et C répondent aux invitations envoyées
3. B répond à l’invitation
4. C répond à l’invitation

Pour ce faire la XEP 0249 « Direct MUC invitations » a été implémentée. Cette extension du protocole XMPP utilise la stance « message » pour notifier un utilisateur qu’il est invité à participer à une conversation. Un utilisateur recevant une invitation peut accepter ou refuser de participer à une conversation.

Exemple d’invitation :

```
<message from='a@conference.xmpp.fr' to='b@conference.xmpp.fr'>

      <x xmlns='jabber:x:conference'
            jid='room@conference.xmpp.fr'
            reason=' Because I'm worth it !'/>

</message>
```

Indicateurs de frappe

Afin de dynamiser les échanges textuels, la XEP 0085 « Chat states notifications » a été implémentée. En étendant des stances « messages », il est possible de renseigner ses interlocuteurs sur son état lors d’un échange textuel. La XEP 0085 recommande l'utilisation de la stance « message » pour signaler ces informations, sous la forme suivante :

```
<message
    from='bernardo@shakespeare.lit/pda'
    to='francisco@shakespeare.lit/elsinore'
    type='chat'>

  <composing xmlns='http://jabber.org/protocol/chatstates'/>

</message>
```

Lorsque qu’un utilisateur interagit avec la zone de composition, une stance « message » renseignant son état est envoyée à intervalle de temps régulier, dans la limite d’un nombre d’envoi fixé.

Etherpad-Lite

Etherpad-Lite est une solution libre d’édition collaborative de documents texte. Basé sur NodeJS, le projet est disponible sous licence APL 2.0. Dans l’optique d’améliorer les interactions entre les utilisateurs pour le travail, le projet Etherpad est apparu comme l’opportunité d’intégrer facilement au système de messagerie un moyen pour les utilisateurs de travailler sur du texte en temps réel : rédaction de messages, listes, …

Les documents Etherpad sont pour l’instant disponibles à titre d’essai, et sont présentés comme des documents éphémères. L’impact de cette solution sera mesuré grâce aux outils statistiques pour déterminer son utilité réelle. Cependant, étant donné la polyvalence de cet outil et sa popularité (notamment grâce au Framapad de Framasoft), il est très probable que cette solution plaise.

Utilisation

Un document Etherpad peut être ouvert de plusieurs manières :

- **Dans le menu** : un utilisateur peu saisir un nom arbitraire de document, et inviter optionnellement des utilisateurs à partager ce document. Les utilisateurs invités peuvent accepter ou refuser de participer à un document.
- **Dans une conversation multi-utilisateurs** : chaque conversation est associée à un document Etherpad potentiel. Ce qui permet à tous les participants d’une conversation de partager le même document Etherpad.
- **Dans n’importe quelle conversation** : en utilisant dans un message la syntaxe de ressource, un lien vers un document est créé automatiquement. Exemple : ‘++etherpad:nom_du_document’

Intégration

Etherpad a été installé sur le serveur de démonstration et ses fichiers de configuration ont été ajouté au dépôt du projet. La configuration d'Etherpad est claire et simple, le serveur écoute un port local en non-sécurisé et le trafic transite par le proxy TLS mis en place avec le serveur Apache HTTP.

Un système simple d'invitations a été élaboré pour améliorer la coordination des utilisateurs :

```
<message from='remi@im.silverpeas.net/g0akw7k37' xmnls='jabber:client' to='user@domain'>

      <etherpad padid='etherpad-document-id'
            allusers='a@domain, b@domain, c@domain'
            status='invitation'
            id='55cd5290-a323-4509-a9b4-71feb043dc2e'/>

</message>
```

Authentification

Pour l'instant aucun système d'authentification n'a été mis en place. Les utilisateurs sont avertis que les documents Etherpad sont accessibles par tous (à condition de connaître l'identifiant d'un document), mais dans l'avenir si cette fonctionnalité a du succès il sera indispensable de gérer les droits d'accès.

Perspectives

Si le système est populaire, il serait peut être intéressant d'intégrer de la même manière d'autres outils de communication en temps réel comme Ethercalc (tableur collaboratif) ou Etherdraw (dessin collaboratif).

La vidéoconférence

Dans ce rapport le terme « vidéoconférence » sera utilisé dans le cas de communications vidéo entre plus de deux personnes. Dans le cas de vidéoconférences entre deux personnes uniquement, les termes « appels vidéo » seront préférés.

La vidéoconférence a été mise en place à titre expérimental afin d'analyser le potentiel de cette fonctionnalité. La vidéoconférence permet de mettre en relation jusqu'à 6 participants avec des flux audio et vidéo.

Le client JSXC utilise la XEP Jingle et WebRTC pour transmettre des flux audio et vidéo entre clients. L'extension Jingle permet de manipuler des sessions multimédia, c'est à dire de signaler les clients, de négocier les transports et de synchroniser les actions d'une session multimédia. WebRTC permet de capturer les flux audio et vidéo à partir de la webcam ou de l'écran d'un utilisateur, et de demander son transport qui est assuré par le navigateur.

Le module Jingle/WebRTC original de JSXC a été entièrement réécrit pour permettre l'utilisation de plusieurs flux vidéos simultanés. En effet, dans le client original, seul un appel vidéo peut être déclenché à la fois. Aucune XEP ne traitant de la vidéoconférence de manière satisfaisante, une extension « maison » du protocole XMPP a été développée. Le fonctionnement de la vidéo conférence peut paraître trivial mais permet un fonctionnement simple sans infrastructure supplémentaire autre que celle nécessaire aux communications multimédia WebRTC en paire à paire.

Principe

La vidéoconférence s'effectue en paire à paire entre chacun des clients. Chaque client reçoit une invitation de l'initiateur avec la liste de tous les clients participants, et détermine quels clients doivent être contactés pour

que tout le monde reçoive les flux audio et vidéo nécessaires.

Chaque client doit être appelé uniquement lorsqu'il est prêt, c'est à dire lorsque l'utilisateur a accepté la vidéoconférence. Pour illustrer les explications qui suivent, l'exemple suivant sera utilisé : L'utilisateur *b@domain/res* initie une vidéoconférence avec *a@domain/res*, *c@domain/res* et *d@domain/res*.

Déterminer qui appeler

Les échanges de flux sont effectués avec l'extension Jingle. Lorsque qu'une session vidéo Jingle est initiée, un client A propose ses flux audio et vidéo à un client B, qui refuse ou accepte en retournant ses propres flux audio et vidéo. Ainsi pour que A et B soient connectés, il suffit que A contacte B ou le contraire.

Afin que chaque client reçoive un flux de chaque participant, mais en évitant l'ouverture de sessions inutiles, une procédure est utilisée pour déterminer quel client doit appeler quel autre client. Dans le cas du client B souhaitant initier une vidéoconférence avec les clients A, C, et D, chaque client reçoit une invitation contenant ces informations:

- b@domain/res: Initiateur
- a@domain/res: Participant
- c@domain/res: Participant
- d@domain/res: Participant

Pour déterminer quels clients il doit contacter, chaque client doit ensuite:

- Créer une liste avec l'initiateur et tous les participants,
- Trier la liste par ordre alphabétique,
- Puis doubler cette liste

Ce qui donne dans notre exemple:

```
[   a@domain/res,
    b@domain/res, // initiateur
    c@domain/res,
    d@domain/res,
    a@domain/res,
    b@domain/res, // initiateur
    c@domain/res,
    d@domain/res  ]
```

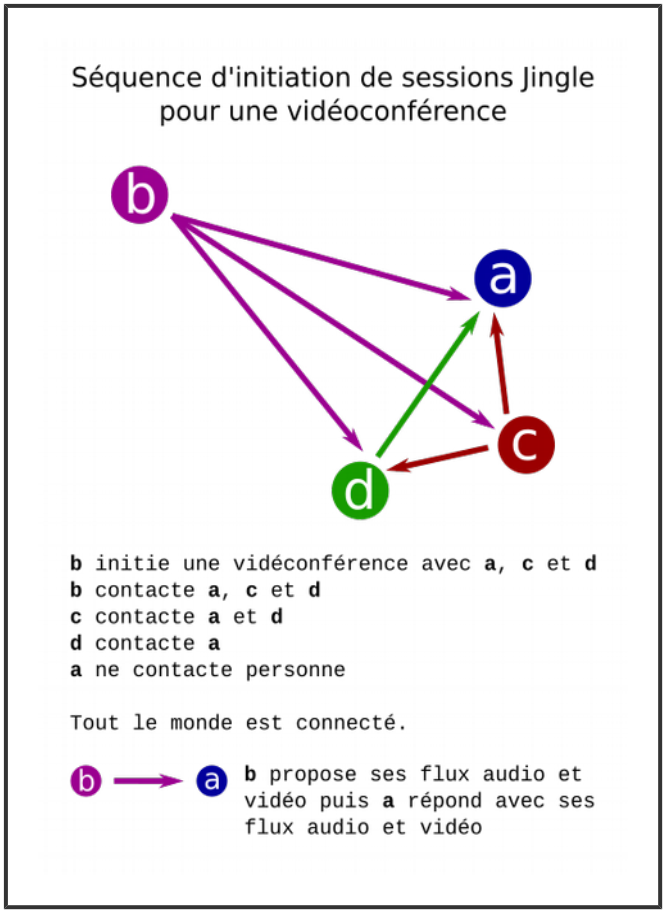
A partir de cette liste chaque client peut déterminer quels clients il doit contacter :

- L'initiateur contacte tous les clients
- Les participants contactent tous les clients de cette liste situés entre la première occurrence de leur propre identifiant et la prochaine occurrence de l'identifiant de l'initiateur

Ce qui donne pour notre exemple la séquence suivante:

- b@domain/res, l'initiateur, devra contacter `c`, `d` et `a`
- c@domain/res devra contacter `d` et `a`
- d@domain/res devra contacter `a`

- a@domain/res ne devra contacter personne



Messages

Les messages utilisés sont dérivés de la stance XMPP « message » permettant d'envoyer à n'importe quel utilisateur un message texte ou plus de données si nécessaire. Les messages utilisés ci-dessous ont toujours ces caractéristiques:

- Les clients y sont représentés par leurs fulljid (identifiant Jabber complet, avec ressource)
- Les clients sont de deux types:
  - un client est initiateur, il prend l'initiative de la vidéoconférence
  - les autres clients sont participants
- Les listes de participants ne contiennent jamais l'initiateur
- L'identifiant d'une vidéoconférence est unique et transmis dans chaque message
- Chaque message est envoyé à l'initiateur et à tous les participants

Les messages utilisés peuvent paraître verbeux, mais ce format a été privilégié pour faciliter la lisibilité du code. En effet il aurait été possible de placer l'initiateur dans la liste des participants, par exemple en première position, mais le message aurait été plus difficile à lire pour une personne extérieure au système.

Initiation d'une vidéoconférence

Exemple de message envoyé lors de l'initiation d'une vidéoconférence :

```
<message from='b@domain/res'
         id='579f7d5a-a70d-4ed8-a660-7f2f3d2e2696'
         xmlns='jabber:client'
         to='a@domain/res'>

<videoconference users='a@domain/res, c@domain/res, d@domain/res'

                  status='initiate'

                  id='579f7d5a-a70d-4ed8-a660-7f2f3d2e2696'
                  initiator='b@domain/res'
                  datetime='2012-06-22 05:40:06'
                  message='B vous invite à une vidéoconférence avec ...'/>
</message>
```

Abandon d'une vidéoconférence

Exemple de message envoyé lorsque qu'un utilisateur refuse de prendre part à une vidéoconférence :

```
<message from='c@domain/res'
         id='579f7d5a-a70d-4ed8-a660-7f2f3d2e2696'
         xmlns='jabber:client'
         to='a@domain/res'>

<videoconference users='a@domain/res, c@domain/res, d@domain/res'

                  status='abort'

                  id='579f7d5a-a70d-4ed8-a660-7f2f3d2e2696'
                  initiator='b@domain/res'
                  datetime='2012-06-22 05:40:06'
                  message='B est occupé ...'/>
</message>
```

Lorsque qu'un message d'abandon est envoyé, tous les participants doivent abandonner la vidéoconférence.

Acceptation de la vidéoconférence

Exemple de message envoyé lorsqu'un participant de la vidéoconférence se déclare prêt à être appelé :

```
<message from='b@domain/res'
         id='579f7d5a-a70d-4ed8-a660-7f2f3d2e2696'
         xmlns='jabber:client'
         to='a@domain/res'>

<videoconference users='a@domain/res, c@domain/res, d@domain/res'

                  status='accepted'

                  id='579f7d5a-a70d-4ed8-a660-7f2f3d2e2696'
                  initiator='b@domain/res'
                  datetime='2012-06-22 05:40:06' />
</message>
```

Lorsque qu'un client reçoit un message d'acceptation, il doit vérifier s'il doit appeler le client émetteur du message.

Ré-invitation dans une vidéoconférence

Les connexions WebRTC pouvant parfois être instables, il est important de laisser l'opportunité aux participants de ré-inviter un participant perdu (ou l'initiateur). Exemple de message de ré-invitation d'un participant perdu :

```
<message from='b@domain/res'
         id='579f7d5a-a70d-4ed8-a660-7f2f3d2e2696'
         xmlns='jabber:client'
         to='a@domain/res'>

<videoconference users='a@domain/res, c@domain/res, d@domain/res'

                  status='reinvitation'
                  reinvite='c@domain/res'

                  id='579f7d5a-a70d-4ed8-a660-7f2f3d2e2696'
                  initiator='b@domain/res'
                  datetime='2012-06-22 05:40:06' />

</message>
```

A la réception de ce message :

- Si le client est la cible de la ré-invitation il doit confirmer son acceptation en envoyant un nouveau message d'acceptation de vidéoconférence
- Si le client est appelant de la cible, à l'arrivée de la confirmation précédemment mentionnée il devra interrompre toutes les sessions Jingle en cours et les renouveler

Module de tests unitaires

Un module de tests unitaires a été ajouté à l'occasion du développement de la vidéoconférence.

Efficacité de la solution

Dans le cas de la vidéoconférence (>2 participants) cette implémentation n'est pas optimale, mais elle a permis des premières manipulations de flux audio et vidéo multiples. Il est recommandé d'utiliser des ordinateurs puissants, une connexion et un navigateur Web rapides. En effet le système dans son état actuel produit des images qui peuvent manquer de fluidité, et le son est régulièrement décalé ou perturbé puisque :

- le transfert des flux dupliqués utilise un grand volume de bande passante
- le traitement des flux par le navigateur est gourmand en ressources (affichage et redimensionnement), et tous les navigateurs n'utilisent pas l'accélération matérielle lorsqu'elle est disponible

Ces problèmes ne concernent que la vidéoconférence (> 2 participants), les appels vidéos eux sont fluides et de bonne qualité.

Un moyen d'améliorer significativement la qualité et la fluidité du système pourrait être l'utilisation d'un serveur MCU (Multi Control Unit). Ce logiciel serveur peut transformer des flux vidéo à la volée et

transmettre des flux allégés, et réduire significativement le nombre de connexions par client.

Partage d'écran

Le partage d'écran permet de diffuser à partir du client JSXC modifié un flux vidéo d'une fenêtre ou de son écran vers un ou plusieurs autres clients. Cette fonctionnalité n'est pour l'instant disponible que sur Chromium ou Chrome, et nécessite l'installation d'une extension. La fonctionnalité sera bientôt disponible sur Firefox.

Cette fonctionnalité est expérimentale et son impact sur l'expérience utilisateur sera mesurée à l'aide des outils statistiques.

Extension Chromium / Chrome

L'utilisation d'une extension pour capturer le flux vidéo de l'écran est indispensable car l'accès au flux d'un écran ne peux se faire qu'à partir du contexte Javascript d'une extension. Pour accéder au flux d'un écran, le client JSXC doit demander le flux à l'extension Chromium / Chrome, qui seule peut exécuter ce code dans son script d'« arrière plan » :

```
/**
 * Listen messages from content script
 */
chrome.runtime.onConnect.addListener(function(port) {
  port.onMessage.addListener(function(message) {

    // someone request screen
    if (message === messages.getScreenSourceId) {
      chrome.desktopCapture.chooseDesktopMedia(screenOptions, port.sender.tab,
             onAccessApproved);
    }

    ...

  });

});
```

Une extension a été développée pour le client JSXC à partir du travail de https://www.webrtc-experiment.com/. La structure de l'extension et la journalisation ont été améliorés, et l'extension a été adaptée à JSXC.

Code source de l'extension :

https://github.com/remipassmoilesel/djoe/tree/master/var/www/djoe/screen-capture

Intégration au portail Silverpeas

L'intégration du système de messagerie au portail Silverpeas est structurée en deux axes :

- **Coté serveur,** avec la répercussion de la création/suppression de comptes et de relations de contact, et le dépôt des informations de connexion au serveur XMPP sur la page
- **Coté client,** avec l'intégration du client JSXC à l'interface existante, l'authentification automatique des utilisateurs et la création d'un module d'interface de programmation applicative

Coté serveur

Répercussion des modifications de comptes utilisateur

La messagerie a été intégrée à la version 6 du portail Silverpeas qui utilise un serveur Java Edition Entreprise Wildfly, la version communautaire du serveur JBoss. Le serveur XMPP utilise sa propre base de donnée d'utilisateurs, et la plateforme Silverpeas la sienne. Du code a été développé pour répercuter sur les deux systèmes :

- La création et la suppression de comptes
- La création et la suppression de relations de contact

Le contrôle du serveur XMPP se fait grâce à l'interface de programmation applicative REST du serveur Openfire. Afin de minimiser l'impact d'un changement de logiciel serveur, la manipulation de cette API a été encapsulée dans une interface nommée « ChatServer ». Dans ce cas précis, l'utilisation d'une API spécifique et non du XMPP a été indispensable. Des travaux peuvent être entrepris pour synchroniser les comptes à l'aide d'annuaires LDAP mais cette approche pourrait manquer de souplesse en cas de mutualisation de serveurs XMPP pour plusieurs organismes distincts.

La répercussion de ces opérations entre le serveur XMPP et le portail Silverpeas a été grandement facilitée par la mise en place d'un bus d'écoute d'événements par l'équipe de développement. Ce bus utilise la gestion des événements de Java EE et permet à un objet d'écouter un type d'événement en étendant simplement une classe, sans enregistrement explicite de l'objet auprès de la source de l'événement.

Dans l'exemple de l'écoute de la création / suppression d'utilisateurs, il suffit d'étendre la classe CDIResourceEventListener<UserEvent> pour déclencher les actions désirées :

```
/**
 * Listen user modifications to clone them in Chat server
 *
 * @author remipassmoilesel
 */
public class UserSubscriptionListener extends CDIResourceEventListener<UserEvent> {

  @Inject
  private ChatServer server;

  @Override
  public void onCreation(final UserEvent event) throws Exception {

       ... server.createUser(full);

  }

}
```

Dépôt des informations de connexion au serveur XMPP

La connexion du client JSXC au serveur XMPP est effectuée lorsque la page Web servie par la plateforme Silverpeas est prête. Les informations suivantes sont nécessaires pour se connecter :

```
userLogin:           Le login de l'utilisateur
userPassword:        Le mot de passe XMPP de l'utilisateur
httpBindUrl:         L'URL de la connexion BOSH (XMPP à travers HTTP)
silverpeasContext:   Le contexte d'URL du portail Silverpeas. Exemple : /silverpeas
```

Ces informations sont ensuite traitées dans la redéfinition de la fonction de connexion du module JSXC/Silverpeas :

```
jsxc.api.registerCallbacks({

      /**
       * Reconnect client on demand from user
       */
      "onReconnectRequest" : function() {
        jsxc.api.Silverpeas.connect();
      },

};
```

Arborescence du code d'intégration :

```
.
├── ChatServerException.java
├── ChatUtils.java
├── HttpRequestHandler.java
├── HttpRequestResponse.java
├── listeners
│   ├── RelationShipListener.java
│   └── UserSubscriptionListener.java
└── servers
    ├── ChatServer.java
    └── OpenfireServer.java
```

Branche d'intégration de la messagerie sur Github :

https://github.com/SilverTeamWork/Silverpeas-Core/tree/chat-integration

Package java/org/silverpeas/core/chat/ :

https://github.com/SilverTeamWork/Silverpeas-Core/tree/chat-integration/core-library/src/main/java/org/silverpeas/core/chat

Coté client

Empaquetage du client

L'empaquetage du client JSXC a été amélioré pour permettre une bonne intégration :

- en réduisant les dépendances du client et en réduisant le code d'intégration
- en isolant les dépendances qui entrent en conflit avec celles utilisées sur la plateforme (notamment i18n)

L'intégration a été facilitée par le travail des développeurs de la plateforme Silverpeas qui ont dans la même période arrêté d'utiliser les « frameset ».

Authentification

L'authentification du client JSXC au serveur XMPP se fait grâce aux informations de connexion déposées par le serveur sur la page, une fois que la page est chargée.

Interface de programmation applicative

Afin de faciliter une meilleure intégration, un module dédié d'interface de programmation applicative à été créé. Ce module permet :

1. De manipuler le client JSXC avec un minimum de connaissances sur son fonctionnement et un risque minimum de levée d'exception. Par exemple, pour lancer une conversation à partir d'une liste d'identifiants :

```
jsxc.api.createNewConversationWith([‘a@domain/res’,’b@domain/res’,’c@domain/res’]) ;
```

2. D'ajouter des modules pour étendre et adapter les possibilités de JSXC à la plateforme cible. Par exemple pour ouvrir une fenêtre de discussion avec un utilisateur à partir d'un identifiant non-XMPP :

```
var SilverpeasCustomModule = {....} ;
jsxc.api.registerCustomModule({
  name : "Silverpeas", module : SilverpeasCustomModule
});

jsxc.api.Silverpeas.openChatWindowById(3) ;   // ‘3’ est un identifiant d’utilisateur
                                              // interne à la plateforme cible
```

3. D'enregistrer des fonctions à l'écoute de certains événements pour adapter le comportement du client JSXC ou de la plateforme cible. Par exemple pour répercuter des relations de contact entre les deux systèmes :

```
var silverpeasCallbackSet = {

  /**
   * Reconnect client on demand from user
   */
  "onReconnectRequest" : function() {
    jsxc.api.Silverpeas.connect();
  },

  /**
   * Send Silverpeas invitation on XMPP buddy added
   *
   * @param buddyBJid
   */
  "onBuddyAdded" : function(buddyBJid) {
    var login = Strophe.getNodeFromJid(buddyBJid);
    jsxc.api.Silverpeas.inviteUser(login);
  },

  "onBuddyAccepted" : function(buddyBJid) {
    jsxc.debug("onBuddyAccepted", buddyBJid);
  }

};

jsxc.api.registerCallbacks(silverpeasCallbackSet);
```

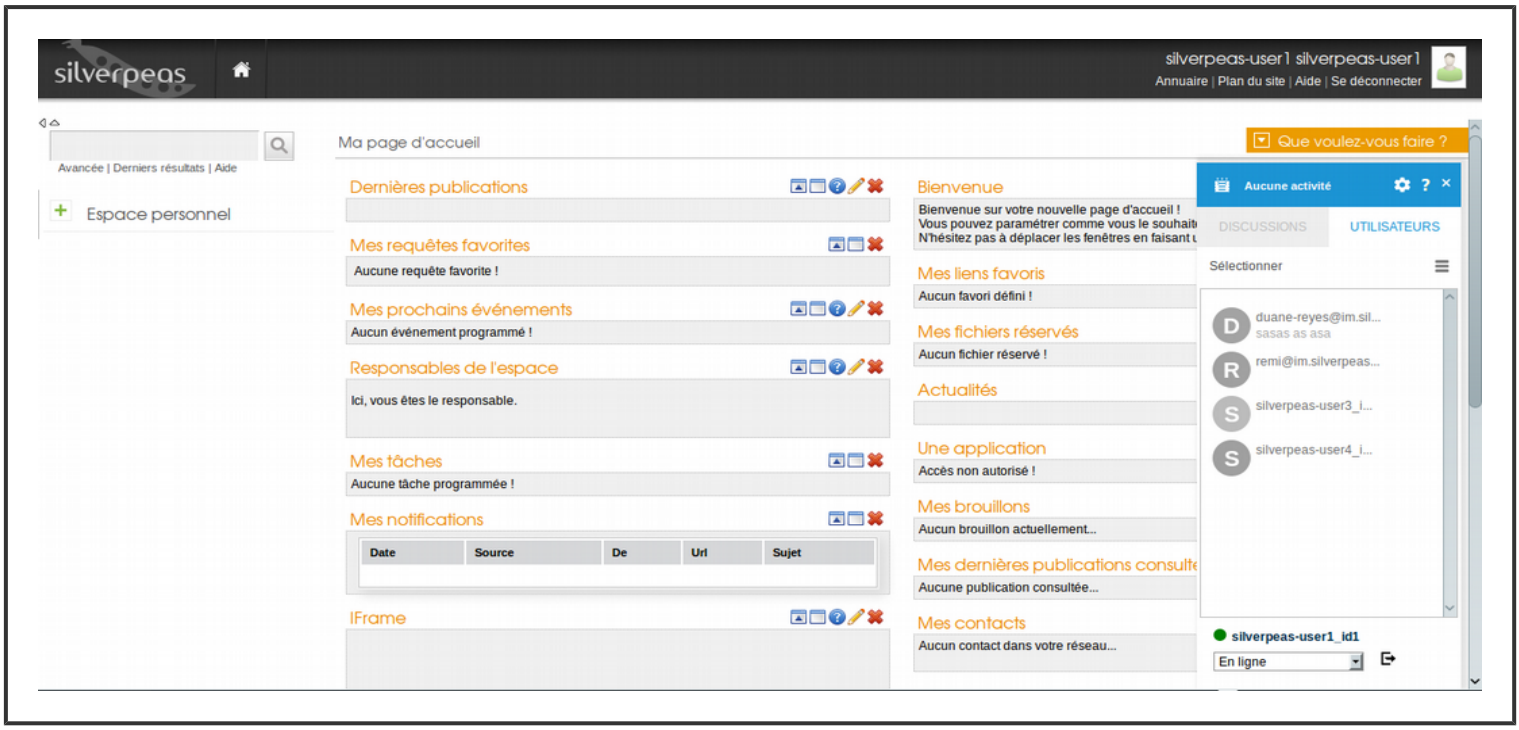
Conflits de dépendances

Le projet JSXC original utilise Grunt et Concat pour extraire les sources de plusieurs fichiers et les concaténer à la suite les unes des autres en scripts ou en styles. Cet empaquetage implique l'appel de dépendances à l'aide de variables Javascript globales (accessibles depuis l'objet *window*), ce qui peut provoquer des conflits si deux dépendances portent le même nom sur une même page.

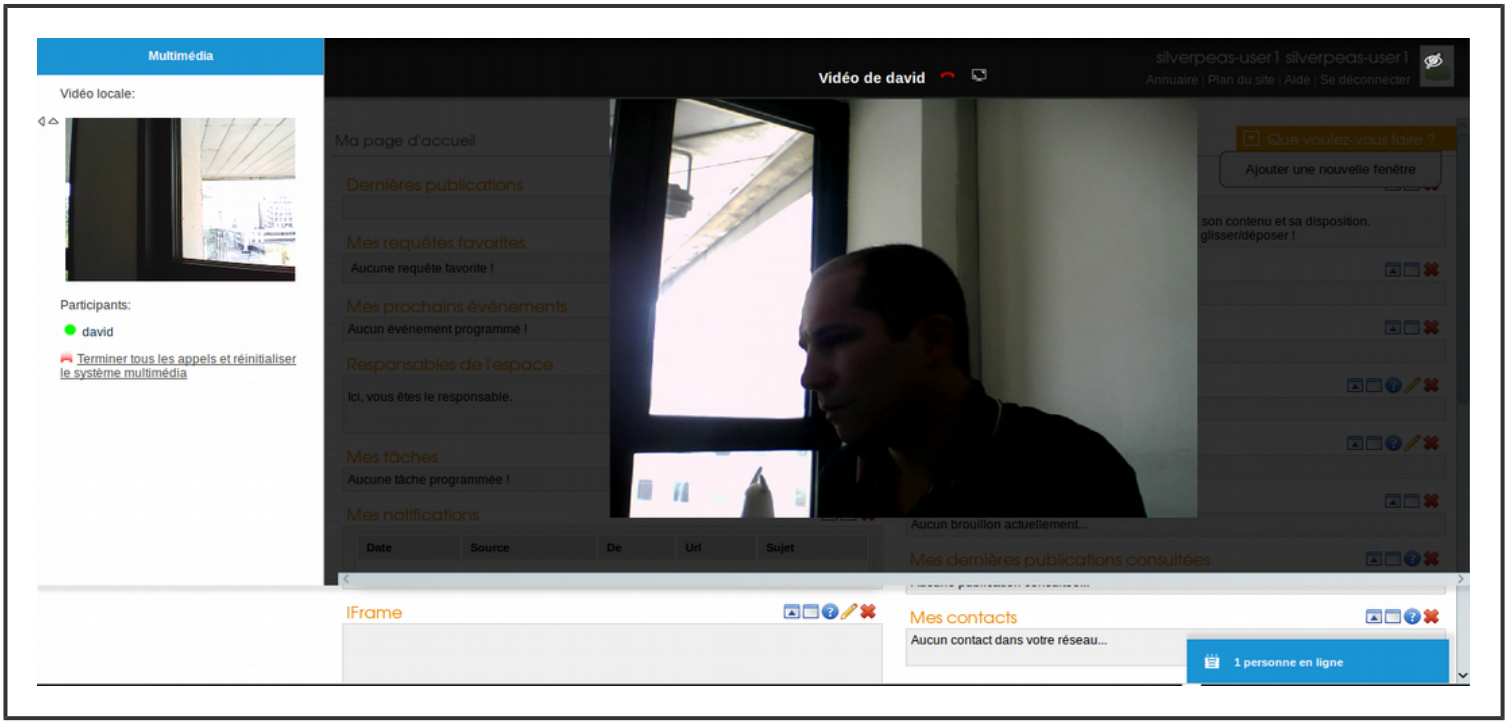
Lors de l'intégration du client JSXC à la plateforme Silverpeas, le plugin JQuery I18n d'internationalisation de JSXC est entré en conflit avec celui de la plateforme Silverpeas (modifié pour les besoins de l'entreprise). Pour pallier à ce problème, un traitement Webpack a été ajouté à la procédure d'empaquetage de JSXC. Désormais, JSXC utilise son propre plugin i18n, référencé dans un objet non global.

L'utilisation de Webpack pour gérer toutes les dépendances du projet serait bénéfique pour mieux isoler le client JSXC, mais cela nécessiterai beaucoup de travail car de très nombreuses références globales sont utilisées.

Vues de l'interface intégrée au portail



La barre de conversation sur la plateforme Silverpeas V6



Un appel vidéo sur la plateforme Silverpeas V6

7. Ce qu'il reste à faire

Ce projet demandera encore beaucoup de travail, notamment pour :

- Fiabiliser toutes les fonctionnalités et généraliser les tests unitaires
- Perfectionner le support des différents navigateurs et systèmes : Internet Explorer, Safari, … et particularités des systèmes mobiles
- Retravailler la structure du client JSXC et purger les éléments obsolètes ou non utilisés
- Améliorer l'image Docker qui permettra de manipuler l'infrastructure logicielle serveur simplement
- Poursuivre l'intégration de la messagerie sur le portail Silverpeas
- Préférer l'utilisation du transport WebSocket quand ce sera possible, et n'utiliser BOSH qu'en transport de secours
- Perfectionner la gestion des historiques de conversation
- (Ré-) Internationaliser le client JSXC
- Créer les DTD nécessaires aux messages XMPP personnalisés (vidéoconférence, Etherpad, …)

8. Conclusion et bilan sur les technologies employées

La réalisation de ce projet de système de communications instantanées a permis la manipulation de technologies actuelles et puissantes, sur des systèmes ouverts : technologies Web, XMPP, WebRTC, systèmes GNU/Linux, … Un des grands intérêts de ce genre de projets est la polyvalence nécessaire pour mener un projet à terme : étude, administration système, développement, ergonomie, …

Le protocole XMPP est parfaitement adapté aux messageries instantanées et encore actuel. Grâce au choix

assumé d’ouverture des protocoles et à leur standardisation, un grand nombre de projets de qualité professionnelle et de documentations sont disponibles. Ce qui permet avec un investissement très limité d’arriver rapidement à un projet de qualité.

WebRTC se révèle fiable et de bonne qualité, mais l’aspect encore expérimental de cette API et le peu de contrôle possible sur les transports utilisés en font un pari risqué en tant que moteur principal d’une messagerie. Cependant, en tant que complément cette technologie permet de proposer des services valorisables qui étaient jusqu’alors inaccessibles sans investissements lourds.

L’objectif de création d’une messagerie texte pour la plateforme Silverpeas a été largement dépassé en terme de fonctionnalités. Cependant, ce système nécessitera encore du travail de fiabilisation pour arriver à un niveau de qualité suffisant pour être distribué.

Cette expérience est prometteuse, et si certaines fonctionnalités peuvent paraître superflues, grâce aux données collectées par le module de statistiques les efforts seront répartis là où ils seront nécessaires. De cette manière l’innovation et le travail de développement suivront les besoins réels des utilisateurs.

Logiciel libre

Si tout ce travail a été rendu accessible à une seule personne en un temps raisonnable, c'est grâce au travail de nombreux acteurs du monde numérique qui choisissent de publier le résultat de leurs travaux sous licence libre.

Le choix de publier des travaux sous licence libre n'est pas qu'un choix éthique engagé, c'est avant tout une méthode de travail efficace. Publier sous licence libre, c'est un moyen d'améliorer la qualité des logiciels produits, un excellent moyen de promouvoir le travail d'une personne ou d'une organisation, et surtout c'est donner à toute une communauté l'opportunité de s'améliorer et de proposer des services toujours plus modernes.

Remerciements

Merci à Miguel Moquillon pour son aide et ses conseils précieux,

Merci à Aurore Allibe pour la conception de l’interface,

Merci à toute l’équipe de l’entreprise Silverpeas.