XMPP au service du système d'information

Jérôme Bousquié
IUT de Rodez
33, avenue du 8 mai 1945 – 12000 Rodez

Standardisé par l'IETF et supporté depuis des années par la XMPP Standards Foundation, le protocole XMPP, Extensible Messaging and Presence Protocol, est initialement dédié à des usages de messagerie instantanée typiquement entre individus.

Cependant ses caractéristiques (sécurité, extensibilité, etc) lui confèrent un intérêt particulier dans le cadre du déploiement d'une architecture d'échange de messages entre des services ou des serveurs du SI.

En effet, non lié à une couche applicative, à un langage de programmation ou un protocole de transport tiers, comme peuvent l'être XML-RPC, JMS, SOAP ou encore REST, XMPP permet à des services de différents niveaux, par exemple, une application de gestion, un service réseau et un serveur LDAP, de communiquer entre-eux.
On parle alors de MOM ou Message Oriented Middleware.

L'exposé vise à présenter de façon générale le protocole XMPP sans s'attarder sur l'aspect du clavardage, mais en mettant plutôt l'accent sur ses potentialités dans le cadre d'une architecture de type MOM.

Au travers d'exemples de déploiements à l'IUT de Rodez, il cherche à montrer d'une part la simplicité et le coût réduit de ce type de déploiement dans un environnement hétérogène et, d'autre part, les capacités de XMPP à répondre à des besoins fréquents comme le traitement asynchrone d'une demande immédiate ou le transport d'un flux continu d'informations.

Mots-clefs : messagerie instantanée, protocole XMPP, architecture événementielle, communication inter-services/serveurs

1 Introduction

Les systèmes d'information des établissement d'enseignement supérieur de recherche comprennent généralement des entités très variées :

- des briques applicatives ou applications de gestion pour, par exemple, la gestion des ressources humaines, la gestion financière et comptable, celle de la paye, de la scolarité, des heures et emplois du temps, des immobilisations, etc
- des éléments d'infrastructure logicielle comme des bases de données intermédiaires, des annuaires (LDAP), des mécanismes de certification et d'authentification (PKI, Radius, Kerberos, CAS, Shibboleth), des services transversaux de bas niveau (DNS, DHCP, etc) et des outils d'administration ou de surveillance (logs, nagios, système de sauvegarde, de load balancing, de virtualisation, antivirus)
- des services à l'utilisateur comme le mail, l'accès au web, au wifi, à un ENT, à un espace de stockage personnel, à des applications web ou tout simplement à des PC de

l'établissement. Ces services s'appuient généralement sur les éléments d'infrastructure précédemment décrits et sont provisionnés par les données venant des briques applicatives.

Aussi la transmission d'informations entre ces entités devient-elle une problématique centrale d'autant que ces entités ne sont habituellement pas pourvues de moyen de communication vers l'extérieur. Elles sont de plus fortement hétérogènes, écrites dans des langages différents, tournant parfois sur des plateformes différentes (OS, JVM) et supportant des formats de données différents.

D'ordinaire la réponse apportée à ces besoins de communication entre les entités du SI varie selon le degré d'homogénéité des entités concernées :

- utilisation d'un protocole ou d'un format natif commun, exemple : ESB, RMI, JMS, JSON
- utilisation de protocoles ou de formats partagés, exemple : XMP-RPC, SOAP ou déclinaison REST
- utilisation de « moulinettes » dédiées d'export/import de fichiers et de formats spécifiques, exemple : synchronisations batch des SGBD vers LDAP, ETL

L'idée est donc de proposer une alternative complémentaire à ces solutions, légère, peu coûteuse à mettre en œuvre et peu intrusive sur l'existant, par le déploiement d'une architecture à base de *Message Oriented Middleware* ou MOM. Le protocole XMPP permet d'implémenter simplement ce type d'architecture.

2 Le protocole XMPP

2.1 Présentation générale

XMPP signifie eXtensible Messaging and Presence Protocol.

Héritier direct du protocole Jabber qui a vu le jour en 1999, conçu par Jeremie Miller, il a la vocation initiale de proposer des fonctionnalités d'échange de messages instantanés et de notification de présence, ou plus généralement d'un statut, entre des utilisateurs connectés au service. Cependant, cet article ne s'attarde pas sur les usages du clavardage entre utilisateurs. Il met plutôt l'accent sur l'utilisation du protocole comme moyen de transport de messages entre programmes informatiques.

XMPP un protocole ouvert de l'internet : il est défini actuellement par les RFC 6120 [1][4], 6121 [2][5] et 6122 [3][6], qui remplacent, pour les deux premières, les RFC 3920 et 3921. Cette définition est complétée par les RFC 3923, 4854, 4979 et 5122 pour des considérations aussi variées que le chiffrement de bout en bout, les URN des espaces de noms, les enregistrements IANA et le schéma d'une URI XMPP.

XMPP est composé de spécifications principales, le Core, et d'extensions nommées XEP pour XMPP Extension Protocol. Seul le Core fait l'objet d'une RFC. Les XEP sont proposés par l'organisme qui supporte et promeut l'usage de XMPP, à savoir la XSF, XMPP Standards Fundation (http://xmpp.org/). Cette fondation est d'ailleurs à l'origine de la rédaction des derniers RFC.

Il s'agit d'un protocole client-serveur à authentification. Le trafic est transporté par TCP sur les ports standards 5222 entre le client et le serveur et 5269 entre serveurs. XMPP est un protocole décentralisé. Les flux sont chiffrés par TLS/SASL.

2.2 Le JID

Chaque utilisateur est identifié par un identifiant unique nommé JID (Jabber ID). Le JID se construit sous la forme :

[nœud "@"] domaine_xmpp ["/" ressource]

exemple : jerome.bousquie@iutrodez.fr/Bureau

Le nœud et la ressource sont des éléments facultatifs. Évidemment, il n'est pas aisé de communiquer avec jerome.bousquie du domaine iutrodez.fr si l'on omet de spécifier le nœud. La ressource permet d'envoyer un message à une destination choisie si le correspondant est connecté plusieurs fois avec le même identifiant de domaine.

Exemple : jerome.bousquie@iutrodez./Bureau et jerome.bousquie@iutrodez.fr/Maison

Dans le cas où la ressource n'est pas spécifiée, le message sera délivré à la connexion de plus haute priorité, propriété de la connexion au moment de son établissement, d'une valeur choisie par le client.

Le domaine est un domaine XMPP. Il s'agit en pratique d'un FQDN qui sera résolu par des enregistrements de type SRV du DNS. Une adresse IP est aussi acceptée. Un même domaine XMPP peut ainsi être géré par plusieurs serveurs, tout comme un serveur peut bien sûr gérer plusieurs domaines XMPP.

2.3 La session

Quand un utilisateur avec le JID toto@domaine1 souhaite envoyer un message au JID titi@domaine2, la séquence suivante se produit :

- toto@domaine1 se connecte (tcp port 5222) et s'authentifie sur un des serveurs XMPP gérant le domaine domaine1
- toto@domaine1 envoie une strophe XML (cf 2.4) à ce serveur stipulant qu'il veut délivrer un message à titi@domaine2
- le serveur gérant domaine1 consulte le DNS gérant la zone domaine2, puis contacte le serveur domaine2 (tcp port 5269) et lui délivre le message immédiatement
- le serveur gérant domaine2 délivre immédiatement le message à l'utilisateur connecté avec le JID titi@domaine2.

Si ce dernier n'est pas connecté, le serveur peut le stocker pour le délivrer dès la prochaine connexion de titi@domaine2. On parle de mode *push* : le client ne requiert pas les messages, c'est le serveur qui les lui envoie. Ce type de fonctionnement n'est évidemment possible que parce qu'une session XMPP consiste en une connexion TCP persistante. C'est le client qui établit la session et généralement lui qui y met fin. La majorité des clients implémentent d'ailleurs une méthode de reconnexion automatique en cas d'interruption accidentelle de la session TCP.

2.4 La strophe

XMPP est un protocole texte. Ce texte, en Unicode encodé en UTF-8, respecte la grammaire XML. Les messages sont structurés en strophes (*stanzas*) qui sont des blocs cohérents du protocole, comme par exemple un ensemble : ordre, arguments, données. Les strophes sont organisées dans un flux de données (*stream*), l'ensemble constituant un seul et même grand document XML.

Il existe essentiellement trois types de strophes : *message*, *presence* et *info/query*. *Message* permet de transporter un message, *presence* sert à notifier un statut, et *info/query* à envoyer une interrogation. Dans la pratique *info/query* sert aussi à presque toute autre type d'action non traité par les types *message* ou *presence*. Seules les strophes de type *message* nous intéresserons dans le cadre de cet article.

Exemple de strophe de type *message* :

```
<message to='romeo@example.net' from='juliet@example.com/balcony' type='chat' xml:lang='en'>
 <body>Wherefore art thou, Romeo?</body>
</message>
```

2.5 Les serveurs et les clients

XMPP n'est pas un protocole jeune. Par ailleurs, il a été déployé et éprouvé à l'échelle de l'internet soit sur des services en ligne gratuits comme Jabber.org ou Googletalk, soit sur la multitude de serveurs décentralisés opérés par des particuliers ou des organisations.

Il existe donc aujourd'hui de très nombreux serveurs et clients XMPP extrêmement matures.

On pourra citer du côté des serveurs open-source :

- Ejabberd qui fait office de référence à ce jour, écrit en erlang réputé pour ses mécanismes de tolérance aux pannes,
- Djabberd, écrit entièrement en perl, apprécié des amateurs de ce langage de scripts pour sa modularité,
- Openfire, écrit en java, et dont une adaptation interne constitue le moteur du service Googletalk,

mais aussi Apache ActiveMQ, un projet de la fondation Apache, ou encore Tigase. La liste n'est pas exhaustive. L'installation et le paramétrage de ces serveurs sont très bien documentés. Il existe même pour certains des assistants de mise en route qui simplifient la configuration initiale.

Du côté client, il y a pléthore de logiciels et ceci sur les principales plate-formes actuelles : Windows, Mac OS, Linux et désormais Android ou iPhone.

On peut citer, en vrac : Psi, Pandion, Gajim, Coccinella, GoogleTalk, Adium, iChat, Pidgim, Spark. La liste est évidemment non complète. La plupart des logiciels de messageries instantanées proposés d'office avec les versions des systèmes d'exploitation sont multi-protocoles et proposent, moyennant quelques paramétrages, de configurer un compte XMPP.

2.6 Les librairies clientes

La maturité du protocole permet de disposer à ce jour d'une grande variété de librairies clientes et ceci dans la plupart des langages. Une librairie cliente XMPP est un ensemble de programmes écrits dans un langage donné qui donne accès à des fonctions, méthodes ou objets, selon le paradigme du langage en question, permettant de réaliser simplement les opérations du protocole en masquant ce dernier.

On pourra donc écrire un programme qui se connecte à un serveur XMPP avec un JID donné, qui envoie ou qui reçoit des messages, qui réagit et répond à ces messages, etc. Un tel programme est appelé un bot XMPP.

Il existe des librairies pour les langages C/C++, java, C#, perl, python, php, ruby, ada, lisp, actionscript, javascript, lua, erlang, haskell, objective-C. Ici encore la liste n'est pas exhaustive et il existe parfois plusieurs librairies différentes pour le même langage.

L'intérêt de disposer d'une telle richesse de librairies est que l'on peut aisément coder des bots sur des machines différentes sans bouleverser l'écosystème.

Exemple : sur un serveur exécutant une application de gestion métier en java, il peut être immédiat de déployer un petit bot tournant sur la même JVM. Il n'est par contre pas judicieux de déployer une JVM sur un serveur Apache ou Squid par exemple, alors qu'un bot en C, voire en perl ou python s'ils sont nativement installés et utilisés sur la machine, sera peu intrusif. De la même façon, sur un serveur Windows Active Directory, pourquoi ne pas utiliser directement du code natif en C# ?

L'idéal est donc d'utiliser au cas par cas une librairie d'un des langages natifs du système (C/C++, C#, ou langages de script par exemple) ou inter-agissant nativement avec le service installé que l'on veut rendre communicant (java sur une JVM, PHP sur une web app PHP, par exemple). Un bot écrit selon ces principes se révèle efficace en terme d'échanges avec le service concerné, peu intrusif en terme de maintenance du code général de la plate-forme et très économe en ressources (CPU, RAM, TCP).

3 Une architecture à base de MOM

3.1 Message Oriented Middleware

Nous venons de voir qu'il est aisé de disposer d'un serveur XMPP interne et qu'il est possible de coder des bots sur la plupart des plate-formes de notre système. L'idée est donc d'utiliser ces bots comme intergiciel (*middleware*) de communication entre les services ou applications de notre SI : les services seront capables d'échanger des messages entre eux de façon instantanée. On parle alors d'architecture à base de Message Oriented Middleware ou MOM.

3.2 Exemple : IUT de Rodez

Un petite architecture à base de MOM a été déployée de la sorte :

- un serveur XMPP Openfire a été installé et gère un domaine XMPP interne nommé iut.rdz, uniquement accessible à la zone des serveurs,
- deux bots ont été codés, l'un sur une machine windows virtuelle du parc enregistrée dans le domaine AD, nommons le bot windows, l'autre sur le serveur proxy web de l'établissement que nous nommerons bot squid.

Le bot windows, codé en C# avec la librairie AGSXMPP [7], est constitué de deux programmes. Le premier programme (ecoute.exe) se connecte avec son JID, récupère en boucle les messages XMPP qui lui sont envoyés et les retourne simplement sur la sortie standard. Le second programme (parle.exe), se connecte de la même façon avec son JID, puis lit en boucle chaque ligne de l'entrée standard qu'il envoie sous forme de message XMPP à un destinataire donné. Ces deux petits programmes, une fois écrits, sont donc grandement réutilisables sur toute machine Windows.

Il suffit alors donc de coder dans un programme externe ou un script, appelons le traite.exe, ce qu'on attend de cette machine. Traite.exe va, en boucle, lire l'entrée standard, effectuer le

traitement demandé selon le message reçu et produire son résultat sur la sortie standard. Un simple enchaînement de pipelines nous permet alors de rendre cette machine communicante sur le service souhaité :

```
ecoute.exe | traite.exe | parle.exe
```

A l'IUT de Rodez, traite.exe réalise, entre autres, la tâche suivante : il reçoit en entrée le nom d'une machine windows du parc et exécute, comme administrateur du domaine, la commande `tasklist /V /S nom_du_pc_distant` qui renvoie la liste des processus en cours sur le PC distant interrogé. Cette liste filtrée et triée est envoyée ligne à ligne au JID destinataire défini dans parle.exe via le pipeline. Cette information mise en forme pourra ensuite être affichée dans une page web d'une application de surveillance et d'administration des machines du parc étudiant.

Cette façon de coder tient du couplage faible : il y a le moins de dépendances possible entre les processus assurant le transport, la réception ou l'envoi d'un message, et le processus de traitement proprement dit. Ce dernier pourrait donc faire partie d'un service ou d'une application déjà en fonction sur la plate-forme.

L'intérêt ici est de pouvoir gérer une demande asynchrone : la demande de la liste des tâches en cours d'un PC distant est transmise et prise en compte immédiatement, mais son traitement et la production du résultat peuvent prendre dans certains cas un temps conséquent. Ce délai, initialement inconnu, est difficilement intégrable dans une solution de type webservice reposant sur un serveur web où la réponse est attendue dans la même transaction http (requête/réponse). Par ailleurs, une solution de type webservice aurait requis d'installer a minima un serveur web sur la machine virtuelle Windows afin de pouvoir écouter les demandes entrantes, solution qui devient vite lourde si l'on multiplie le nombre des machines communicantes.

Le deuxième bot, squid, codé en ruby avec la librairie xmpp4r [8], est chargé d'exécuter à la demande un équivalent de la commande bash `tail -f /var/log/squid3/access.log | grep pattern`, c'est à dire de lire en direct le log de Squid contenant un motif donné. Il envoie chaque ligne produite à un JID destinataire à la volée. Ruby n'a été choisi que parce qu'il était déjà déployé et utilisé sur le serveur hébergeant Squid et pour sa concision. Un autre choix de langage natif aurait pu être aussi judicieux (C, python, perl). Ce bot démarre ou arrête la lecture du fichier de log lorsqu'il en reçoit l'ordre par un message XMPP. Il illustre un autre usage du MOM, c'est à dire l'envoi immédiat et continu d'un flux de données, opération ici encore plus difficilement réalisable avec une solution de type webservice.

Le flux envoyé est relayé dans une page web HTML5 ayant ouvert une websocket sur un serveur de websockets. Le fonctionnement de websocket, dont la RFC est encore à l'état de brouillon [9], ne sera pas détaillé dans cet article. Il s'agit, pour résumer, d'une connexion tcp permanente entre un navigateur web et le serveur de websocket permettant à ce dernier de faire du *push* de données vers le navigateur quand il le souhaite. Un programme en javascript se charge alors de traiter les données provenant du serveur, de les mettre en forme et de les afficher.

On voit ici ce qu'une architecture événementielle à base d'intergiciel orienté message permet de réaliser plus facilement qu'avec des technologies plus transactionnelles dans lesquelles la réponse est associée à la requête :

- la possibilité de procéder facilement à des traitements asynchrones : une demande est produite immédiatement auprès d'un service, mais celui-ci peut décider quand et en combien de temps il répondra. On trouvera une illustration de cette technique de délégation de tâches dans cette présentation : [10].

– La possibilité de communiquer des flux d'informations en temps réel pendant un temps donné en s'appuyant sur l'aspect « instantanéité » du protocole.

4 Pour aller plus loin

4.1 PubSub

Dans les exemples précédents, chaque JID recevait des messages ou en envoyait directement à un autre JID. C'est une communication de point à point. On comprend aisément que dans une architecture plus complexe qui comprendrait de nombreux bots, il deviendrait vite fastidieux de gérer qui doit notifier quoi à qui.

Un serveur XMPP implémentant la XEP 0060 [11] permet de proposer le mécanisme PubSub [12] ou Publish/Subscribe. Un JID émetteur de messages peut alors notifier, non plus un autre JID d'une information, mais un nœud de publication déclaré sur le serveur. Il faut donc voir un nœud PubSub comme un canal identifié sur un sujet donné. Les JIDs intéressés par ce type d'informations de leur côté souscrivent à la connexion à ce nœud. Les messages envoyés sur le nœud leur sont alors immédiatement délivrés.

Exemple : on pourrait imaginer qu'un bot associé à l'application de gestion du personnel envoie sur un canal PubSub nommé GRH la notification qu'un nouveau personnel a été saisi dans la base de données. Un bot associé au référentiel de comptes (annuaire LDAP) et inscrit au canal GRH pourrait par exemple être notifié et déclencher une création de compte, un autre bot aussi inscrit au canal GRH et associé à l'application de paye pourrait créer une instance de la personne dans sa base et un troisième bot, toujours inscrit au canal GRH, et déployé sur le système ToIP pourrait inscrire la personne dans l'annuaire téléphonique.

Un serveur de messages instantanés fonctionnant sur le mode de canaux de distribution ou de queues de messages est appelé *message broke*r (traduction maladroite : courtier de messages) dans une architecture à base de MOM. Nous conserverons dans la suite l'appellation anglaise de *broker*.

4.2 AMQP

XMPP n'est pas le seul protocole ouvert de messagerie instantanée pouvant être utilisé dans une architecture à base de MOM. Les deux plus aboutis sont STOMP [13] pour Simple Text Oriented Messaging Protocol et AMQP [14] pour Advanced Message Queuing Protocol.

Comme XMPP, STOMP est un protocole texte. Il hérite grossièrement de la forme du protocole HTTP. Son fonctionnement nominal correspond à la description de PubSub, c'est à dire qu'un serveur STOMP est généralement un *broker* de messages. Bien que STOMP soit maintenant mature, il ne dispose pas comme XMPP d'une offre de clients et de serveurs aussi large et n'a pas été éprouvé dans la réalité à l'échelle de l'internet. La documentation y est aussi plus pauvre. Enfin, ses spécifications ne font pas l'objet de RFC et ne sont maintenues que par la communauté STOMP. Il reste néanmoins un protocole simple et efficace à l'échelle d'une organisation interne.

AMQP est le projet le plus abouti de MOM ouvert pour le monde de l'entreprise. Les spécifications d'AMQP ne font pas non plus l'objet de RFC, mais le protocole est porté par un consortium d'acteurs majeurs de l'informatique et du secteur bancaire. On citera en autres : Cisco, Microsoft, Novell, Red Hat, Software AG, VM Ware, Bank of America, Barclays Bank, Credit Suisse, Goldman Sachs, JPMorgan Chase Bank. Leur but est de parvenir à un standard sécurisé, fiable et réellement interopérable (plus que l'approche JMS).

Ainsi AMQP est un protocole de transfert de messages d'entreprise. Ce n'est pas un protocole texte, les données sont échangées en binaire sous forme de flux d'octets. En cela, il autorise

bien mieux que XMPP le transfert de fichiers par exemple. Un serveur AMQP fonctionne de facto comme un *broker* de messages organisé sur un modèle de queues et de routage : les messages sont émis sur des canaux, mais des règles (conformité du contenu d'un message à un modèle, par exemple) permettent au serveur de réaffecter dynamiquement la distribution à d'autres canaux. AMQP est très mature et utilisé dans le monde de l'entreprise, en particulier dans le secteur bancaire comme outil d'interopérabilité. Ses spécifications sont bien documentées. Elles sont maintenues uniquement par le consortium. On trouve une offre de serveurs et de librairies clientes, certes pour l'instant moins riche que celle de XMPP, mais en constante augmentation.

5 Conclusion

Nous venons de voir qu'une architecture à base de MOM pouvait à moindres coûts d'installation, de codage, de maintenance et d'exploitation compléter les technologies de communication ou d'interopérabilité déjà déployées dans un SI. Elle peut apporter une réactivité supplémentaire dans le système (instantanéité), autoriser la délégation de tâches à des processus à forts délais, permettre de diffuser des flux de données ou simplement, par sa non-dépendance à un langage ou à un OS, rendre des services hétérogènes communicants.

XMPP est une solution simple et légère à mettre en œuvre. La grande variété des librairies et la documentation très riche permettent de procéder à un déploiement rapide sur les SI les plus hétérogènes tant que les besoins ne dépassent pas les possibilités de PubSub. Pour des besoins plus complexes comme le routage dynamique de messages, il sera nécessaire de quitter l'univers des RFC pour celui des standards de l'entreprise et de passer à AMQP.

6 Bibliographie

[1] http://www.rfc-editor.org/rfc/rfc6120.txt

[2] http://www.rfc-editor.org/rfc/rfc6121.txt

[3] http://www.rfc-editor.org/rfc/rfc6122.txt

[4] http://www.bortzmeyer.org/6120.html

[5] http://www.bortzmeyer.org/6121.html

[6] http://www.bortzmeyer.org/6122.html

[7] http://www.ag-software.de/agsxmpp-sdk/

[8] http://home.gna.org/xmpp4r/

[9] http://datatracker.ietf.org/doc/draft-ietf-hybi-thewebsocketprotocol/

[10]http://www.igvita.com/2009/04/06/henry-ford-event-driven-architecture/

[11]http://xmpp.org/extensions/xep-0060.html

[12]http://www.igniterealtime.org/support/articles/pubsub.jsp

[13]http://stomp.github.com//

[14]http://www.amqp.org/