Propagation d'événements entre passerelles OSGi

Didier Donsez
Laboratoire LSR, Equipe ADELE
Université Joseph Fourier
BP 53, F38041 Grenoble Cedex 9,
didier.donsez@imag.fr

Gaël Thomas
Laboratoire LSR, Equipe ADELE
Université Joseph Fourier
BP 53, F38041 Grenoble Cedex 9,
gael.thomas@imag.fr

RESUME
Le service de communication événementielle d'OSGi offre un cadre standard pour faire communiquer des services co-localisés. Nous proposons de propager ces événements hors de la passerelle à l'aide de ponts qui ne nécessitent aucune modification ni chez les émetteurs/récepteurs, ni dans le service d'événements. Plusieurs ponts peuvent alors cohabiter au sein de la même passerelle ce qui masque l'hétérogénéité des intergiciels à l'application et fait inter-opérer ces intergiciels. Notre infrastructure permet la construction de nouvelles applications reposant sur des intergiciels orientés message et nécessitant la flexibilité apportée par OSGi. Trois implantations de ponts sont présentés dans cet article et valident notre approche.

MOTS CLES
OSGi, Evénement, Publication-Souscription, Pont

1. INTRODUCTION
Le service de communication événementiel Event Admin (chapitre 113) dans la spécification R4 d'OSGi [8] offre un cadre uniforme de communication par messages entre services au sein d'une passerelle centralisée. Le service EventAdmin est un médiateur d'événements entre des rédacteurs (*publisher*) et des souscripteurs (*subscriber*) collocalisés sur le même canevas OSGi. Dans cet article, nous nous sommes intéressés à rompre les frontières de la passerelle pour propager les événements vers d'autres passerelles OSGi et vers et depuis d'autres canevas (J2EE, application Flash, …) au travers de MOM (Message Oriented Middleware) [4] aux propriétés diverses dépendant de l'environnement d'exécution (réseau ad-hoc, peer-to-peer, grande échelle…). La flexibilité inhérente à OSGi et la communication répartie par message sont un besoin pour construire de nouvelles applications flexibles orientées message dont la liste, non exhaustive, est :

- Médiateurs M2M [1][2][7][16] (RFID, Capteurs) dont les événements RFID sont collectés, agrégés, corrélés depuis les capteurs/lecteurs jusqu'aux serveurs IT des entreprises;
- Enterprise Service Bus (ESB) [17] qui se définit comme l'architecture orientée service (SOA) pilotée par les événements dans lequel de nouveaux services peuvent être intégrés dynamiquement;
- Applications collaboratives [3] dans lesquelles les actions réalisées dans l'espace de travail d'un collaborateur sont traduites sous la forme d'événements qui sont propagés vers les autres espaces de travail des autres collaborateurs;
- Canevas orientés événements (DVB-MHP, [10]…) dans lesquels les activités sont déclenchées par les actions de l'usager ou par les événements provenant du réseau de diffusion;
- Gestionnaires autonomiques [6] qui sont construits sur la base de la boucle autonomique dans laquelle les sondes remontent mesures et alertes sous la forme d'événements.

Notre approche repose sur la notion de pont. Un pont EventAdmin est un émetteur/récepteur qui s'occupe de propager les événements en dehors de la passerelle de manière transparente pour les rédacteurs et pour les souscripteurs. La notion de pont permet de séparer totalement les rédacteurs/souscripteurs OSGi des protocoles utilisés pour faire communiquer les passerelles. Plusieurs ponts peuvent aussi cohabiter et utiliser différents protocoles (JMS, Ivy, GENA, Scribe…) : l'EventAdmin sert alors de pivot. La sémantique de l'EventAdmin reste inchangée et les rédacteurs/souscripteurs écrits pour l'EventAdmin n'ont pas besoin d'être modifiés lors de l'introduction de nouveaux ponts. Les deux principales contributions de nos travaux sont :

- La séparation entre les applications utilisant le service d'événements d'OSGi et le MOM, ce qui augmente la réutilisabilité des clients de l'EventAdmin;
- L'utilisation du service d'événements d'OSGi pour convertir des messages provenant de MOM différents, ce qui diminue l'hétérogénéité entre MOM.

Cet article est structuré de la façon suivante. La section 2 détaille le service d'événements d'OSGi qui sert de base à nos travaux. La section 3 décrit les principes et problèmes de la réalisation d'un pont entre OSGi et un protocole particulier. La section 4 décrit trois réalisations de ponts qui valident notre approche : un pont avec JMS (Joram), un pont avec Ivy et un pont avec des applications Flash. La section 5 décrit des travaux similaires au notre et enfin la section 6 présente une conclusion et des perspectives de nos travaux.

2. Le service EventAdmin
Jusqu'à la version R3 de la spécification OSGi, les bundles interagissaient soit via les appels de méthode des services, soit via les Wires du WireAdmin selon le modèle Producteur-Consommateur. La version R4 [8] a introduit le service EventAdmin (chapitre 113) qui offre un modèle de communication événementiel entre les bundles par publication-souscription [4]. L'objet événement (Event) est associé à un sujet (topic) et qualifié par des propriétés. L'éditeur poste (i.e. publie) un événement au service EventAdmin qui le diffuse (potentiellement en parallèle) à tous les souscripteurs du sujet. La figure 1 schématise le fonctionnement. Chaque souscripteur doit enregistrer un service EventHandler. Ce service doit être qualifié par une liste de sujets et de patrons de sujets (propriété *eventadmin.topics*) auxquels le souscripteur s'abonne. Comme les

sujets sont organisés de manière hiérarchique, un patron de sujets permet de souscrire à tous les événements publiés sur un sujet et les sous-sujets dans la hiérarchie (cas du troisième souscripteur de la figure 1 qui souscrit à la fois au sujet « tic », au sujet « tic/tac » et à tous ses sous-sujets). La qualification peut être complétée par une expression de filtre (propriété *eventadmin.filter*) sur les événements pour limiter les événements que souhaite traiter le souscripteur. La publication synchrone (méthode *sendEvent(Event)*) d'un événement est synchronisée sur la terminaison des exécutions de tous les services de réception (EventHandler) concernés. La publication asynchrone (méthode *postEvent(Event)*) libère immédiatement l'éditeur. Le service EventAdmin garantit l'ordre des événements. Le service EventAdmin assure une livraison causale des événements et gère une liste noire des EventHandler défectueux ou consommant trop de CPU.

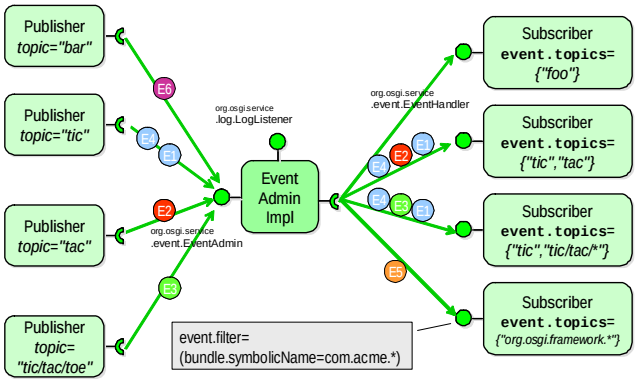

Figure 1. Exemple de publication-souscription.

Ce service est en réalité central dans la spécification car beaucoup d'autres services (UPnP, Configuration, Log…) doivent désormais publier sous la forme d'événements EventAdmin les modifications qui se produisent en leur sein. C'est d'ailleurs le cas des événements du noyau (core) : la spécification définit une liste de sujets prédéfinis qui sont liés aux événements du cycle de vie des services, des bundles et du framework.

Les communications de type publication-souscription entre bundles OSGi ne sont pas nouvelles [10] néanmoins la standardisation du service EventAdmin donne de nouvelles opportunités d'applications pour la passerelle OSGi.

3. PRINCIPES ET PROBLEMES

Un pont entre l'EventAdmin et un MOM particulier ne requiert pas de modification dans le code du service implémentant l'EventAdmin : un pont est un émetteur/récepteur comme un autre qui capture tous les événements reçus par l'EventAdmin. Le pont convertit les événements de l'EventAdmin pour le MOM et les propage. De la même façon, un message reçu via le MOM est converti puis envoyé vers l'EventAdmin. La figure 2 décrit un exemple de pont. Le pont EA-MOM capture les événements de l'EventAdmin et les propage via l'implémentation du MOM (flèche descendante) vers les autres passerelles, et reçoit les messages du MOM et les propagent dans l'EventAdmin.

Trois problèmes principaux doivent être résolus lors de la conception d'un pont : (i) quels événements de l'EventAdmin sont propagés et comment sont convertis les événements de l'EventAdmin en messages pour le MOM, (ii) comment sont associés les topics de l'EventAdmin aux canaux de communication du MOM et comment sont créés, trouvés et détruits ces canaux, (iii) comment sont respectés la causalité et la synchronisation.

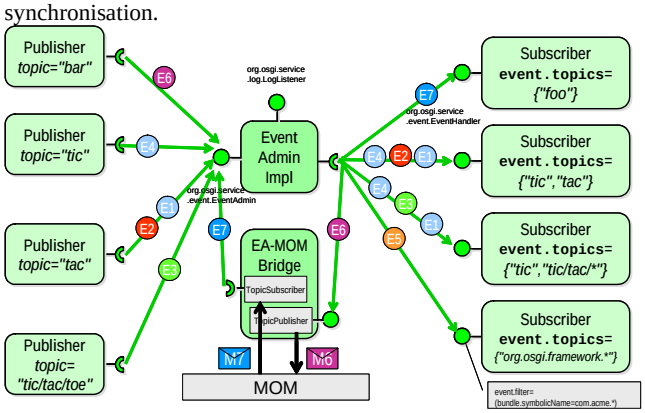

Figure 2. Exemple de pont EventAdmin avec un MOM.

3.1 Politique de propagation

3.1.1 Les données qui peuvent être propagées

Tous les événements de l'EventAdmin ne peuvent pas être propagés car les données associées à un événement peuvent ne pas avoir de sens sur une autre passerelle (i.e. ces données ne sont pas sérialisables). Deux catégories de données non sérialisables sont à prendre en compte : les données dépendant de la machine (socket, fichiers ouverts…) et les données dépendant de la passerelle OSGi (Bundle, bundleID, ServiceReference, …). Les données de la première catégorie ne peuvent pas être converties par le pont car le pont ne peut pas connaître la sémantique de transfert attendue par le l'application. En revanche, la seconde catégorie peut être sérialisée. Un bundle, par exemple peut être sérialisé sous la forme d'un bundleID et un bundleID sous la forme d'une structure comprenant la machine émettrice et l'identifiant du bundle associé. Toutefois, quelque soit la catégorie, le souscripteur distant (i.e. EventHandler) du message doit être en mesure d'interpréter le format modifié de l'événement.

Dans les différents exemples de pont présentés, nous avons choisi de laisser à la charge d'un service tiers le soin de convertir les données non sérialisables.

3.1.2 Choix des données à propager

Tous les événements ne pouvant être propagés, il faut que le pont puisse sélectionner quels événements doivent être propagés. Deux approches permettent de faire cette sélection : propager systématiquement les événements reçus sur un sujet (topic) particulier et choisir une propriété de l'événement qui spécifie si l'événement doit être propagé. La première technique ne nécessite pas de modification des souscripteurs alors que la seconde nécessite une modification pour ajouter la propriété de propagation. Dans un souci de réutilisabilité, nous avons opté, dans nos exemples, pour la première solution. Sélectionner quels sujets seront propagés par le pont nécessite une entité tierce d'administration. Cette entité s'occupe aussi de sérialiser les objets que le pont ne peut pas convertir.

3.2 Sujets et canaux de communication

Pour des MOM possédant différents canaux de communication, ce qui est le cas notamment de JMS dans lequel un canal de communication est représenté par un *topic* JMS, il devient intéressant pour le pont d'associer les sujets de l'EventAdmin avec des canaux particulier. En effet, plusieurs canaux de communications permettent d'éviter d'inonder le réseau et les passerelles avec des événements qui ne les intéressent pas. Le pont d'une passerelle qui ne possède pas de souscripteur pour un sujet donné ne va pas s'abonner au canal associé au *topic* : le pont ne recevra donc que les événements qui l'intéresse.

Le pont pour un MOM qui possède plusieurs canaux doit donc prendre en compte l'association sujet EventAdmin - canal de communication du MOM. Cette association consiste en (i) un mécanisme de cherche du canal de communications associés au sujet, (ii) créer et détruire les canaux de communications.

Il faut noter que l'EventAdmin ne permet pas de déclarer les émetteurs : seuls les récepteurs sont déclarés dans l'EventAdmin. Si un sujet ne possède plus de récepteur, il n'est plus nécessaire de propager les événements et le canal de communication associé au sujet peut alors être supprimé. Toutefois, le moyen de connaître les récepteurs reliés à l'EventAdmin n'est pas spécifié et nous n'avons donc pas implanté le mécanisme de destruction de canal de communication.

3.3 Causalité et synchronisation

Le MOM doit être capable de respecter la causalité des messages pour respecter la spécification de l'EventAdmin. Parmi nos trois exemples de ponts, seul Ivy ne respecte pas cette causalité.

La publication d'un événement peut également se faire de manière synchrone avec l'appel *sendEvent(Event).* Les ponts doivent donc prendre en compte ce besoin. La version courante de nos prototypes de ponts ne respecte pas les émissions synchrones.

4. PONTS DISPONIBLES

Trois réalisations de ponts ont été effectuées pour valider nos travaux. Ces réalisations mettent en avant les deux objectifs des ponts : l'augmentation de la réutilisabilité car les récepteurs et l'EventAdmin n'ont pas besoin d'être réécrits, et la diminution de l'hétérogénité car les trois ponts peuvent cohabiter simultanément sur une seule passerelle ce qui permet de faire interagir différents MOM ensemble.

4.1 Ivy

Ivy est un bus à message qui permet à des applications (C, C++, Java, Perl sur Unix,Windows et MacOS) de diffuser des messages texte sur un réseau local ou sur une adresse multicast. Ivy qui se base sur UDP, ne garantit pas la délivrance ou la causalité des messages. Le mécanisme de souscription est basé sur les expressions régulières vérifiant le contenu textuel des messages diffusés. Ivy s'adresse donc aux applications de l'informatique diffuse hors infrastructure (ie ad-hoc), une des cibles privilégiées de la plateforme OSGi. Ce pont permet à des plateformes présentes dans un réseau ad-hoc de se notifier des événements. Les événements sont encodés comme des messages XML canoniques. La souscription d'un EventHandler dans une passerelle se traduit par la souscription par le pont aux messages Ivy correspondant à l'élément <topic> des messages**.** Comme Ivy ne possède qu'un seul canal de communication (l'adresse multicast), le problème d'association entre sujet et canal de communication ne se pose pas.

4.2 JMS

JMS (Java Message Service) est une spécification Java pour de l'envoie de message. La spécification JMS permet d'unifier l'accès à un MOM. Tous comme nos travaux, JMS augmente la réutilisabilité des composants logiciels car les émetteurs et récepteurs sont indépendants des algorithmes utilisés pour implanter le MOM. Toutefois, JMS n'adresse pas le problème de l'hétérogénéité et ne s'occupe pas de faire interagir ensemble des MOM différents.

Contrairement à Ivy qui ne possède qu'un seul canal de communication (ce qui empêche le passage à l'échelle) et qui n'assure pas la causalité des messages, le pont JMS résout ces deux problèmes et permet aussi de faire interagir une plateforme OSGi avec un serveur J2EE. En revanche, JMS n'offre pas d'API pour réaliser des appels synchrones et nous n'avons pas implanté ce mécanisme dans le pont.

La spécification JMS ne décrit pas la partie administration d'un serveur JMS, notre pont JMS est donc partiellement dépendant de l'implantation particulière ObjectWeb Joram, une implantation open-source de JMS développée par Scalagent.

JMS définit la notion de *topics* JMS sur lesquels des émetteurs et des récepteurs peuvent envoyer et recevoir des messages. La notion de *topic* JMS est semblable à la notion de sujet EventAdmin. Nous avons donc choisi d'associer un sujet EventAdmin à un *topic* JMS du même nom (qui est donc un canal de communication). JNDI est utilisé pour localiser les topics JMS. L'EventAdmin permet à un souscripteur de souscrire à plusieurs sujets hiérarchisés (voir section 2). JMS n'offre pas une telle fonctionnalité, en revanche, l'implantation Joram offre une notion similaire avec des arbres de *topics* dans laquelle un *topic* fils reçoit les messages reçus par son père. Les sujets hiérarchiques EventAdmin sont aussi directement projetés vers les topics hiérarchiques Joram.

Lorsqu'un nouveau rédacteur (resp. un nouveau souscripteur) EventAdmin publie (resp. souscrit) sur un sujet, il faut créer le topic JMS associé. Utiliser un serveur Joram centralisé unique limite le passage à l'échelle. Nous avons donc choisi d'instancier plusieurs serveurs Joram sur une partie des passerelles OSGi. La distribution des serveurs Joram impliquent le problème de création concurrente de topics Joram : deux serveurs Joram peuvent tenter de créer en parallèle des topics Joram de même nom. Le service de nommage JNDI est alors utilisé pour éliminer une des deux créations : dans ce contexte, le service de nommage refuse l'un des deux enregistrements et le serveur Joram qui se voit refuser l'enregistrement détruit le topic redondant et renvoie vers le topic enregistré sur le JNDI.

4.3 Flash

La technologie de présentation Flash est largement répandue chez les infographistes pour la réalisation très rapide de présentations dynamiques et interactives. Dans le contexte d'IHM embarquée sur une plateforme OSGi, cette technologie peut fort bien se substituer à des canevas tels que Swing et SWT qui n'ont aucune pénétration dans le monde de l'infographie. Une application Flash qui s'exécute dans sa propre machine virtuelle (à coté de la machine virtuelle Java de la passerelle OSGi) a la possibilité de

dialoguer avec tout autre application sur l'hôte via une socket. Dans ce contexte, il nous a paru intéressant de permettre à une application Flash de souscrire à des événements EventAdmin et de réagir à ceux-ci (par exemple, en affichant un message d'alerte) et d'en émettre à son tour. Le pont entre l'application Flash et l'EventAdmin définit une liste de requêtes (suivant une grammaire XML). Ces requêtes servent à la souscription et au désabonnement aux sujets (topics) des événements manipulés par l'EventAdmin. Elles servent aussi à l'envoi et à la réception des événements EventAdmin par l'application Flash. Ces événements subissent une sérialisation XML assez simple similaire à celle utilisé par le pont Ivy. Le pont Flash est un pont point à point : à chaque instance de pont est associé une machine virtuelle Flash unique : le pont Flash n'est pas à proprement parler un pont vers un MOM. Le problème d'association entre sujet EventAdmin et canal de communication ne se pose donc pas.

5. TRAVAUX RELATIFS

La passerelle OSGi est par essence destinée à communiquer avec son environnement réseau. La première spécification OSGi a introduit un service HTTP permettant à une passerelle d'héberger des sites web dynamiques tandis la troisième spécification a ajouté un pont vers des périphériques JINI et UPnP présents dans le réseau adhoc dans lequel se trouve la passerelle.

Depuis, plusieurs auteurs se sont intéressés à réaliser des ponts de communication vers et pour des services distants. La plupart des travaux [14][15] se sont concentrés sur les communications de type RPC [13] avec des protocoles tels Corba/IIOP, RMI ou JXTA et ont délaissé les communications par passage de messages [4][12] pourtant bien adapter aux applications ciblées par OSGi. La découverte des services s'appuie sur des protocoles ad-hocs comme SLP, DNS-SD, JXTA-DS ou sur des annuaires d'infrastructure comme RMIRegistry, CosNaming, LDAP. R-OSGi [15] annonce pourvoir relayer les événements EventAdmin.

Un travail très similaire au notre mais néanmoins postérieur est le pont entre l'EventAdmin et l'Eclipse Communication Framework (ECF) réalisé par Bob Brady dans le cadre de son Master [3]. L'objectif est de réaliser des applications collaboratives propageant leurs changements d'état selon le modèle de communication publication-souscription.

6. CONCLUSION ET PERSPECTIVES

Cet article présente les dispositifs de propagation d'événements EventAdmin en dehors d'une passerelle OSGi. Cette propagation est transparente aux rédacteurs et aux souscripteurs. L'utilisation de ponts isole les souscripteurs et les rédacteurs de la technologie MOM utilisée pour la propagation. Cette approche augmente la réutilisabilité des modules logiciels développés pour OSGi.

Trois ponts sont actuellement opérationnels et d'autres sont en cours de développement (Siena, GENA (UPnP), JINI Eventing, DPWS [11], JXTA [9], BIP [5] et CORBA CosEvent). Ces ponts sont les points de départ d'une réflexion sur les mécanismes de communication par message entre OSGi. Ces ponts seront proposés à la communauté open-source Apache Felix.

7. REMERCIEMENTS

Ce projet est partiellement financé pour le Ministère de l'Industrie dans le cadre du projet ITEA S4ALL.

8. REFERENCES

[1] Philippe Lalanda. E-Services Infrastructure in Power Distribution. IEEE Internet Computing, May-June 2005.

[2] G. Wiederhold. Mediators in the architecture of future information systems. IEEE Computer, 25(3):3849, 1992.

[3] Bob Brady. Developing Collaborative Tools With Equinox and ECF, .ECLIPSEWORLD 2006, Cambridge, MA, Sept. 8, 2006

[4] Patrick Th. Eugster, Pascal Felber, Rachid Guerraoui, Anne-Marie Kermarrec: The many faces of publish/subscribe. ACM Comput. Surv. 35(2): 114-131 (2003)

[5] Rémi Emonet, Dominique Vaufreydaz, Patrick Reignier, Julien Letessier , "O3MiSCID, a Middleware for Pervasive Environments", 1st IEEE International Workshop on Services Integration in Pervasive Environments, June 29, 2006, Lyon, France

[6] Kephart J.O., Chess D.M., "The Vision of Autonomic Computing", IEEE Computer, Janvier 2003.

[7] International Telecommunication Union, "The Internet of Things, Executive Summary", ITU Internet Reports 2005, November 2005 http://www.itu.int/osg/spu/publications/internetofthings/InternetofThings_summary.pdf

[8] OSGi Alliance, "OSGi Service Platform Specification (4d Release)," Octobre 2005, http://www.osgi.org.

[9] Sous projet JMS for JXTA, http://jms-for-jxta.jxta.org/

[10] OSGi TV

[11] F. Jammes, A. Mensch, H. Smit, "Service-oriented device communications using the devices profile for web services", Proc. *3rd international workshop on Middleware for pervasive and ad-hoc computing*, Grenoble, France, Nov. 2005

[12] Daniel A. Menascé: MOM vs. RPC: Communication Models for Distributed Applications. IEEE Internet Computing 9(2): 90-93 (2005)

[13] A. Birrell and B. Nelson. Implementing remote procedure calls. ACM Trans. on Computer Systems, 2(1):39--59, February 1984

[14] André Bottaro, Anne Gérodolle, Philippe Lalanda, "Pervasive spontaneous composition", 1st IEEE International Workshop on Services Integration in Pervasive Environments, June 29, 2006, Lyon, France.

[15] Jan S. Rellermeyer, "flowSGi: A Framework for Dynamic Fluid Applications", ETH Zurich, Master's Thesis, 2006, http://www.iks.inf.ethz.ch/publications/files/flowSGI.pdf

[16] Colombe Héraut, Gaël Thomas, Philippe Lalanda. Mediation and enterprise service bus - a position paper. Proceedings of the First International Workshop on Mediation in Semantic Web Services, Amsterdam, Netherlands, December 2005.

[17] M.-T. Schmidt, B. Hutchison, P. Lambros, R. Phippen , The Enterprise Service Bus: Making service-oriented architecture real, IBM Systems Journal, Volume 44 Issue 4, 2005, https://www.research.ibm.com/journal/sj/444/schmidt.pdf