Conception d'un service de communication de groupe adaptatif

Hatem Arous, Thierry Villemur, Khalil Drira, Ernesto Expósito

Conception d’un service de communication de groupe adaptatif

Hatem Arous[1, 2], Thierry Villemur[1, 2], Khalil Drira[1, 2] and Ernesto Exposito[1, 2]
1 CNRS ; LAAS ; 7 avenue du colonel Roche, F-31077 Toulouse Cedex 4, France
2 Université de Toulouse ; UPS, INSA, INP, ISAE ; UT1, UTM, LAAS ; F-31077 Toulouse Cedex 4, France
{harous, villemur, drira, ernesto.exposito}@laas.fr

Résumé— **L'analyse des travaux récents autour des services et des plateformes middleware font apparaître trois grandes catégories : Middleware Orienté Message (MOM), Appel de Procédure Distant et Manipulation d'Objet Distant. Notre choix s'est orienté vers le MOM Data Distribution Service (DDS), conçu selon le modèle Publier/Souscrire. En effet, DDS nous semble le mieux adapté pour développer des applications temps réel sensibles au contexte. Nous présentons par la suite notre travail de conception et de mise en œuvre d'un service générique adaptatif pour la communication de groupe au dessus de DDS.**

Mots clés; Service de communication de groupe, Qualité de Service, Middleware, Data Distribution Service, Java Message Service

I. .INTRODUCTION

Les applications distribuées coopératives multiutilisateurs manipulent différents types de médias et impliquent des composants et des services diffus interagissant, selon des schémas de coopération composés, en temps réel.

L'arrivée à maturité des réseaux sans fil préfigure une mobilité généralisée des utilisateurs. Dans le futur, l'information sera diffusée beaucoup plus largement par une utilisation massive d'équipements embarqués : les systèmes de communication seront ubiquistes, mobiles et hétérogènes, autorisant la mise en œuvre d'activités coopératives complexes dans des environnements peu ou pas pré-équipés en infrastructure réseau, typiquement sans fil et ad-hoc. Les applications conçues pour de tels environnements requièrent un niveau élevé de fiabilité et une garantie de la qualité de service.

Dans ce contexte, un enjeu de recherche majeur est de permettre aux applications et à leurs utilisateurs de disposer de services de coopération et de communication de groupe qui offrent des qualités de service adaptées en fonction des ressources machine/réseau disponibles, aux capacités hétérogènes et variables.

L'approche à explorer repose sur l'adaptation dynamique des protocoles de communication de groupe orientés événement (event-based communication) selon le schéma architectural producteur/consommateur (producter /consumer) sur les équipements embarqués.

Cet article est organisé comme suit :

Dans la section II, nous présentons les plateformes middleware ainsi que leurs techniques les plus connues avec des exemples de plateformes middleware. Nous détaillons les middlewares Java Message Service (JMS) et Data Distribution Service (DDS).

Nous présentons aussi, dans la section II, le concept de qualité de service dans les middlewares ainsi que le modèle de communication Publier/Souscrire.

Dans les sections III, IV et V, nous décrivons notre approche et nos travaux réalisés. Une analyse approfondie de la communication de groupe en utilisant les plateformes middlewares a renforcé le choix de DDS pour mettre en place le service de communication.

Dans la section III nous spécifions le service adaptatif pour la communication de groupe. L’architecture du service est présentée dans la section IV. La section V décrit nos travaux et notre scénario d’application.

La dernière section conclue avec les travaux futurs que nous envisageons.

II. ETAT DE L’ART

A. Plateforme middleware

Dans le domaine des systèmes distribués une plateforme middleware est une couche logicielle incluse entre la couche du système d’exploitation et la couche des applications utilisateur de chaque équipement [8].

Une plateforme middleware crée une couche de communication pour l’échange d’informations entre différentes applications. Toutes ces applications sont connectées au middleware de la même façon à travers les composants interface logiciels proposés par la plateforme. Ces composants logiciels gèrent l’hétérogénéité des technologies réseaux mises en jeu.

Les composants logiciels des middlewares assurent la communication entre les applications indépendamment des caractéristiques matérielles et logicielles du terminal de l’utilisateur mis en jeu, des composants logiciels et matériels du réseau, des protocoles réseau et des systèmes d’exploitation…

Trois principales techniques sont utilisées pour l’échange d’informations entre les applications à travers une plateforme middleware :

- Echange de message,

- Appel de procédure distant,
- Et manipulation d'objets distants.

1) Manipulation d'objets distants (ORB - Object Request Broker)

En utilisant un middleware orienté objet qui implémente une interface d'interaction distante, une application peut manipuler des objets distants appartenant à d'autres applications. Les informations reliées aux objets sont éditées et transformées. Le mécanisme de manipulation d'objet est similaire au mécanisme d'appel de procédure distant : les composants du middleware fournissent une souche (stub) dans l'application. Cette souche est un objet spécial qui est relié à l'objet distant. Les opérations exécutées sur l'objet spécial sont automatiquement transmises par la couche middleware à l'application qui possède l'objet distant.

Pour être accessible à distance, les composants du middleware offrent des ressources du côté de l'objet distant (skeleton) qui le permettent. Ce dernier, incorporé dans l'application fournisseur, contient l'objet à manipuler et les interfaces nécessaires pour recevoir et exécuter les demandes distantes. Un skeleton est créé par le biais du mécanisme d'héritage de la programmation orientée objet.

Microsoft DCOM (Distributed Component Object Model) est une technique ORB basée sur le protocole réseau de l'appel de procédure distant.

DCE (Distributed Computing Environment) est un middleware à manipulation d'objet distant proposé par l'OMG (Object Management Group) basé sur CORBA (Common Object Request Broker Architecture).

2) Appel de procédure distant (RPC - Remote Procedural Call)

L'appel de procédure est un mécanisme classique qui autorise l'échange d'informations à travers des paramètres entre un programme et un sous-programme, les deux incorporés dans la même application. L'appel de procédure distant étend ce mécanisme entre un programme et un sous-programme distant accessible à travers le réseau. Grâce au mécanisme RPC du middleware, les fonctions et les procédures implantées dans une application choisie peuvent être accessibles et exécutées à la demande par d'autres applications.

Les composants logiciels du middleware créent, dans l'application source, une souche (stub) qui crée une interface pour l'application destination. L'appel de procédure, fait à travers la souche de l'application source, est transmis par la plateforme middleware à l'application destination où les composants du MW créent une autre souche de réception liée à la souche source.

Les résultats obtenus sont retournés d'une manière symétrique à l'application source. Toutes les informations échangées entre les souches utilisent les techniques de sérialisation offertes par le middleware.

Le protocole RPC de SUN [18] est fréquemment utilisé pour réaliser des appels distants.

SOAP [19] (Simple Object Access Protocol) est une technique pour les appels distants effectués sur les serveurs Web basés sur les protocoles XML (eXtensible Markup Language) et HTTP (HyperText Transfer Protocol).

RMI [20] (Remote Method Invocation) est une technique intermédiaire entre les appels de procédures et la manipulation d'objets distants. C'est un composant logiciel pour exécuter l'appel de procédure distant sur les objets implantés avec le langage de programmation Java.

3) Middleware orienté message (MOM – Message Oriented Middleware)

Les middlewares orientés messages font partie des éléments techniques de base des architectures des ordinateurs. Ils créent un couplage faible entre les applications.

En utilisant un MOM, les applications communiquent en échangeant des messages d'une manière similaire à l'échange des e-mails. Une application source envoie un message. Le message est transmis à l'application destination par la couche middleware en même temps que l'application source fait d'autres traitements (asynchrones).

Le contenu du message suit un format connu à l'avance par l'expéditeur et le destinataire pour pouvoir être traité par l'application réceptrice. Le langage XML est souvent utilisé pour l'écriture des messages échangés.

Un message peut être transmis à une application spécifique choisie par l'expéditeur à partir d'une liste d'applications qui y sont inscrites ou bien à toutes les applications connectées à la couche middleware. Les messages sont triés par ordre de priorité et stockés dans des queues (files d'attentes) par les composants logiciels du middleware.

Java Message Service (JMS) est une API (Application Programming Interface) pour les MOM [2].

Data distribution Service (DDS) [21] est un autre MOM.

Il existe deux modèles principaux pour les MOM :

- Le modèle Point à Point : l'application source produit des messages et l'application destination les consomme. Les messages sont lus uniquement par un seul consommateur. Une fois que le message est consommé, il est supprimé de la file d'attente.
- Le modèle Publier/Souscrire : l'application consommatrice dois s'inscrire à un sujet (Topic) qui définit une catégorie de messages. Les messages sont envoyés au Topic adéquat et restent dans la file d'attente jusqu'à ce qu'ils soient consommés par toutes les applications inscrites.

Le modèle Publier/Souscrire est bien approprié pour les communications de groupe (présentées par la suite). La section suivante détaille ce modèle.

4) Les middlewares Publier/Souscrire :

De nos jours, plusieurs systèmes, en particulier les systèmes distribués, utilisent les middlewares Publier/Souscrire en raison des leurs avantages (performance, rendement, coût, sociabilité, évolutivité, qualité de service (QoS), robustesse…) [3]

Dans les middlewares Publier/Souscrire, les messages sont envoyés en les liant à un Topic. Ce dernier identifie un domaine d'intérêt.

Les consommateurs s'inscrivent au Topic pour lequel ils ont une certaine affinité. Une fois que la souscription est faite, tout message lié au Topic sera automatiquement reçu par le groupe des consommateurs qui y ont souscris. Tous les messages seront reçus par l'ensemble des clients abonnés.

Un message est supprimé de la file d'attente du MW quand tous les consommateurs l'auraient lu et l'auraient acquitté (ou si la date d'expiration du message est atteinte).

Les plateformes middlewares basées sur le modèle de communication Publier/Souscrire les plus utilisées sont Java Message Service (JMS) et Data distribution Service (DDS).

5) Java Message Service (JMS)

JMS est une API standard, proposée par Sun Microsystem [2,4], pour le développement des applications échangeant des messages. L'envoi et la réception des messages sont normalement faits d'une façon asynchrone entre des applications ou des composants Java à travers des files d'attentes (Queues) ou des Topics. D'autre part, les communications synchrones sont permises : dans la communication point à point, quand un client utilise une méthode bloquante, la synchronisation se fait entre le client et le propriétaire de la méthode.

L'API JMS offre des interfaces pour les applications connectées directement aux autres MOMs [1].

Une application cliente utilisant l'API JMS doit exécuter la succession des étapes suivantes pour interagir avec un service de messagerie :

- Localiser le fournisseur de connexion,
- Créer la connexion JMS,
- Créer une session JMS,
- Localiser le destinataire JMS,
- Créer un producteur JMS ou un Consommateur JMS,
- Envoyer ou recevoir un message.

Selon Laranjeiro [1], certaines implémentations de JMS présentent quelques sérieux problèmes de robustesse et de graves problèmes de sécurité.

Wu [2] propose FJM2, un middleware JMS rapide et fiable. Leur système JMS est décentralisé et est basé sur les communications en mode diffusion.

La plupart des implantations de JMS sont centralisées. En conséquence, si le serveur s'arrête, la communication s'arrête aussi. De la même façon, lorsque la taille du message augmente, le temps de latence augmente aussi [4], ce qui cause une incohérence dans un contexte temps réel.

Pour conclure, JMS n'est pas approprié pour les systèmes temps réel et pour les environnements distribués où plusieurs utilisateurs et groupes interagissent par le biais de communications fréquentes.

6) Data Distribution Service (DDS)

DDS pour les systèmes temps-réel [6] est une plateforme de MOM standardisée par l'OMG, qui fournit une technologie avancée pour l'échange de données à travers un réseau. Ce middleware est basé sur le modèle Publier/Souscrire [9].

DDS permet aux applications faiblement couplées d'échanger des informations indépendamment de l'architecture sous-jacente, des langages de programmation et des systèmes d'exploitation [7].

Ce middleware est destiné principalement aux applications industrielles qui requièrent de fortes contraintes en termes de fiabilité et de performance, comme l'aéronautique, la défense ou les télécommunications, et qui incluent la gestion de données complexes. DDS présente ses APIs comme des services pour les applications. La spécification de DDS décrit deux niveaux [5] :

- Un premier niveau (en bas) DCPS (Data-Centric Publish-Subscribe), qui garantie un envoi efficace (envoi de la bonne donnée à la bonne destination). Il est hautement configurable, étroitement lié aux données et possède plusieurs paramètres de QoS pour déterminer le comportement requis.
- Un deuxième niveau plus haut, optionnel, DLRL (Data Local Reconstruction Layer), qui renforce l'intégration de DDS dans la couche application.

DDS supporte le concept de domaine qui regroupe une variété de machines distribuées à travers un réseau et partageant des données communes. Le principe opératoire est un mécanisme de publication et de souscription, permettant aux applications de :

- Modifier la donnée partagée et de notifier par la suite les autres applications par publication,
- S'abonner à la donnée partagée et recevoir les modifications faites par les applications distantes.

DDS est utilisé pour :

- Une diffusion rapide des données vers plusieurs nœuds,
- Les réseaux dynamiques,
- Des exigences de livraison flexible.

Les données sont décrites à travers des langages appropriés tel qu'IDL du standard CORBA.

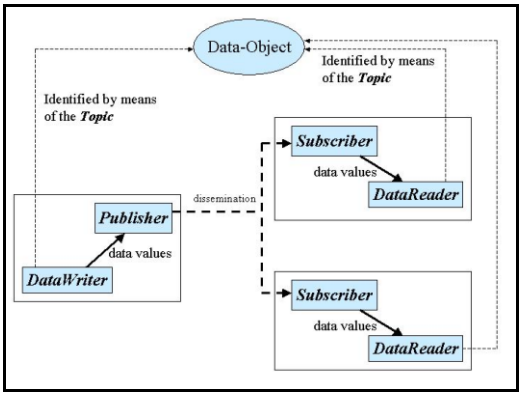

Figure 1. La spécification de DDS [6]

Comme le montre la figure 1, DDS définit plusieurs entités:

- DomainParticipantFactory: le point d'entrée principal de DDS.
- DomainParticipant: le point d'entrée pour la communication dans une zone déterminée.
- TopicDescription: une classe abstraite de base pour les Topic, les MultiTopics et les ContentFilterTopics.
- Topic: la spécialisation de TopicDescription. C'est la description la plus élémentaire des données à publier et à consommer.
- ContentFilterTopic: une spécialisation de TopicDescription (comme Topic) qui permet plus de souscription basée contenu.
- MultiTopic: une spécialisation de TopicDescription comme Topic mais qui permet, en plus, aux consommateurs de combiner, de filtrer et de réarranger les données de différents Topics.
- Publisher: c'est un objet responsable de la distribution des publications.
- DataWriter: permet à l'application de mettre une valeur de donnée publiée dans le Topic.
- Subscriber: c'est un objet responsable de la réception des données provenant du Topic.
- DataReader: permet à l'application de déclarer la données qu'elle souhaite recevoir (par la souscription au Topic approprié) et récupère la valeur de la données.

B. Qualité de service (QoS – Quality of Service)

Dans les systèmes communicants des données, le terme « Qualité de service » se réfère au mécanisme de contrôle de réservations des ressources. La QoS est un ensemble de paramètres qui règle les communications. Il peut aussi gérer les priorités entre différentes applications, différents utilisateurs ou différents flux de données, ou pour maintenir et garantir un certain niveau de performance pour les échanges des flux de données.

Les principaux paramètres de la QoS sont :

- La durabilité : elle indique si la donnée doit être gardée après sa livraison. En général, elle est associée à un service de gestion de durabilité.
- Le temps de latence : il définit le délai maximal acceptable entre la date d'envoi de la donnée et la date de notification de sa réception. Toutes les données envoyées sont copiées dans le cache de l'application réceptrice.
- La fiabilité : elle définit la période d'activité du système d'une manière fiable.
- La gigue (jitter) : elle définit la variation du délai requis par paquet à transmettre de la source à la destination. Cette variation dépend de la position du paquet dans les files d'attente des routeurs tout au long du chemin parcouru entre la source et la destination. La variation de cette position est imprévisible. Une valeur élevée de la gigue peut affecter considérablement la qualité du flux de données, par exemple s'il s'agit d'un streaming audio et/ou vidéo.
- La priorité : quand plusieurs services ou applications se partagent des ressources communes limitées, des priorités sont fixées pour ces services ou applications pour gérer les conflits…

La garantie de la QoS est un enjeu majeur pour les systèmes informatiques de nos jours. La QoS est introduite dans différents domaines et technologies comme le Cloud Computing [13], les réseaux sans fil et les réseaux de capteurs sans fil [12], les systèmes distribués, les réseaux Ad Hoc Mobiles [11] et les services web [14].

DDS offre un niveau élevé et flexible de la gestion de la QoS. Les spécifications de DDS incluent plusieurs paramètres de QoS qui influencent le comportement des services de DDS.

Ces paramètres peuvent être associés à toutes les entités DDS (Publisher, Subscriber, Topic…) [6,7].

Parmi les paramètres qui peuvent être gérées à travers DDS [6]:

- La durabilité,
- Le temps de latence,
- La fiabilité,
- Histoire : spécifie le comportement du service dans le cas ou les valeurs des données changent (une ou plusieurs fois) avant qu'elle ne soit communiquée avec succès à un ou à plusieurs consommateurs existant (garder l'historique des changements pour les transférer plus tard aux nouveaux consommateurs ou bien garder uniquement la dernière mise à jour)

C. Communication de groupe

La gestion de communications entre un groupe de participants est un défi majeur pour garantir un routage correct

et efficace des données à travers les réseaux informatiques [17].

Les travaux récents comme Bruhadeshwar et al. [15] ou Choi et al. [16] s'intéressent au problème de sécurité.

La sécurité peut être une caractéristique supplémentaire de la QoS, pour la communication de groupe à grande échelle, en particulier pour les systèmes temps réel.

Notre contribution consiste à garantir un certain niveau de QoS, choisi en avance par l'utilisateur qui pourrait être mis à jour par la suite, dans la communication de groupe.

III. SPECIFICATION

Dans notre approche nous visons à traiter la communication de groupe. Ce groupe contient plusieurs types d'entités :

- Au sommet de la hiérarchie, un superviseur,
- Dans le niveau suivant on trouve un ou plusieurs coordinateurs,
- Et dans le dernier niveau on trouve les investigateurs.

La figure 2 présent une structure d'architecture hiérarchique typique.

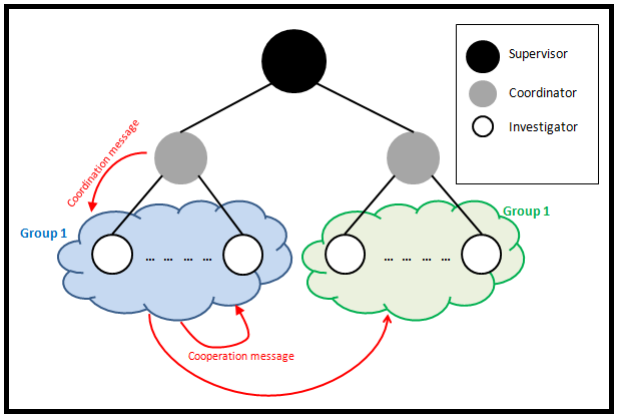

Figure 2. Structure d'architecture hiérarchique

Nous considérons la communication de groupe à grande échelle. Dans ce cas, il est difficile de gérer tous les investigateurs à travers une seule entité centrale qui pourrait se surcharger et s'arrêter en peu de temps dès que le nombre d'investigateurs dépasse quelques milliers. Pour traiter ce problème, nous avons décidé de distribuer la gestion des investigateurs sur différents nœuds. Nous aurons plusieurs nœuds plus stables qui s'occupent de la connexion et déconnexion des investigateurs. Dans ce cas, nous serons aptes à gérer les groupes de communication fortement dynamique.

En conséquence, les investigateurs fortement dynamiques (connexion/déconnexion) vont être gérés par des nœuds plus stables (les coordinateurs). Ces coordinateurs sont eux même gérés par une autre entité centrale stable qui est le superviseur.

Dans une telle architecture, on rencontre deux types de messages à échanger :

- Messages de coordination (verticaux) : ils sont échangés entre des entités de niveaux hiérarchiques différents (du superviseur au coordinateur ou du coordinateur vers l'investigateur)
- Messages de coopération (horizontaux) : ils circulent entre les membres d'un même groupe ou entre les membres de différents groupes, mais dans le même niveau de hiérarchie (par exemple, entre les investigateurs du même groupe).

Dans le contexte temps réel, nous devons préserver la QoS. En effet, une telle architecture doit s'appliquer dans les réseaux locaux standards et aussi dans les réseaux à capteurs par exemple. Dans le cas de milliers d'investigateurs et de centaines de coordinateurs (grande échelle), nous aurons à gérer plusieurs paramètres de QoS prudemment, comme le temps de latence, la fiabilité, la gigue et les priorités, dans le but de ne pas surcharger les réseaux et d'assurer le bon fonctionnement du système.

IV. ARCHITECTURE

Dans les systèmes à grande échelle basés sur l'architecture hiérarchique détaillée dans la section précédente, l'utilisation des middlewares basés sur le modèle Publier/Souscrire (Section II.A) semble être une solution intéressante pour gérer les communications entre les différentes entités qui interviennent.

Le tableau 1 présente une brève comparaison technologique de certains middlewares et de Frameworks.

TABLE I. COMPARAISON DE MIDDLEWARE

Comparison criteria	**Middleware**			
	OSGi	***CCM***	***DDS***	***JMS***
Standard	OSGi Alliance	OMG	OMG	Sun Microsys tems
Download applications in runtime	Y	Y	Y	
Service discovery in runtime	Y	N	Y	N
Manage latency	N	N	Y	N
Platform dependancy	Y	Y	Y	N
Distributed architecture	Y	Y	Y	Y
Network abstraction	N	Y	Y	Y
Supported development language	Java	C/C++, Java	C/C++, Java…	Java
Network bandwith		Low	Low	Low
Robustness			Y	Y
Riliabilty	Y		Y	Y
Communication Model			Point to Point, Multicast and	Point to Point and Multicast

Comparison criteria	Middleware			
	OSGi	*CCM*	*DDS*	*JMS*
			Diffusion	

Nous concluons que le middleware DDS est un bon candidat pour notre approche, vu qu'il répond à nos attentes concernant l'architecture distribuée, l'abstraction du réseau qui généralise son utilisation sur tout type de réseau, l'utilisation du modèle Publier/Souscrire et aussi pour ses capacités de gestion et de configuration des paramètres de la QoS.

V. APPLICATION

Notre objectif est de développer un service de communication de groupe adaptatif et générique qui permet de gérer les entités intervenantes et les messages échangés dans une architecture hiérarchique (section III).

Ce service permettra la gestion de ces entités par :

- L'ajout d'un nouveau coordinateur,
- L'ajout d'un nouvel investigateur,
- Le déploiement des nouvelles entités…

Il permet la gestion des messages en leurs affectant des priorités.

Ce service, nommé Plateforme d'Exécution de Service (PES), utilise le middleware DDS comme le montre la figure 3.

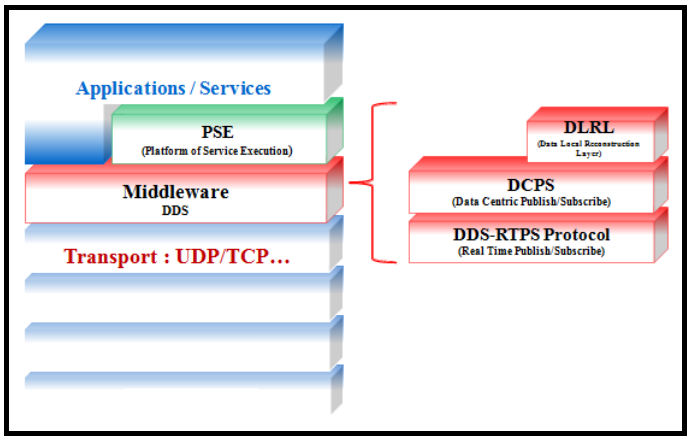

Figure 3. Architecture en couche

Ce travail est fait en partenariat avec d'autres équipes de recherche et des industriels dans le cadre d'un projet national AMIC-TCP (Architecture de Multiplexage Informatique et de Communication pour le Transport en Commun des Personnes).

Le scénario d'application suivant a été proposé pour illustrer notre approche dans le cadre du projet AMIC-TCP.

Nous considérons que la centrale de commande de la flotte des bus est le superviseur responsable de la gestion de tout le système informatique de la flotte. Chaque bus est équipé d'un ordinateur embarqué qui joue le rôle de coordinateur. Ce coordinateur gère les communications entre les passagers dans le bus et les communications avec d'autres passagers dans d'autres bus. Il offre aussi des services tels que le service de dialogue textuel (chat) ou le service de publicité par exemple aux passagers.

Un passager peut utiliser son PDA, son ordinateur portable ou tout autre équipement informatique pour se connecter, charger et utiliser les applications fournies par le coordinateur du bus.

Si une application n'est pas disponible chez le coordinateur, ce dernier peut la charger pour le passager en temps réel à partir du superviseur.

Si un passager souhaite communiquer via une application de "chat" par exemple avec un autre passager dans le même bus, le coordinateur gère cette communication. Mais si le deuxième passager est dans un autre bus, les messages seront routés par le coordinateur du premier bus vers le superviseur, qui, à son tour, les envoie vers le coordinateur du deuxième bus pour qu'ils arrivent à leur destination finale.

Un autre scénario considère deux passagers ou plus qui commencent une communication dans un arrêt de bus par exemple est la poursuivent dans des bus différents. La communication initiale reste toujours active et les utilisateurs restent connectés même dans les bus.

Actuellement les travaux réalisés considèrent une architecture statique dont le nombre d'entités participantes, leurs rôles (superviseur, coordinateur ou investigateur) et leurs positions dans le réseau sont connus à l'avance.

VI. CONCLUSION ET TRAVAUX FUTURS

Nous visons à développer un système de communication de groupe basé sur le middleware DDS pour le domaine des communications coopératives multisessions et multiutilisateurs. Ce système de communication fournit un service de coopération adaptatif et générique.

Dans une prochaine étape, nous nous intéresserons à une architecture dynamique, où de nouveaux coordinateurs ou investigateurs pourront être ajoutés dynamiquement. Nous allons améliorer les techniques de coopération et de coordination inter et intra-groupe basés sur les concepts de Topic et Domain offerts pas DDS.

Ensuite, nous améliorerons notre service pour qu'il assure le déploiement dynamique en temps réel des entités communicantes selon une architecture donnée en entrée au service, architecture décrite selon le format XML.

REFERENCES

[1] N. Laranjeiro , "Experimental Robustness Evaluation for JMS Middleware," IEEE ICSC 2008, in press.

[2] R.-S. Wu, "FJM2 – A Decentralized JMS System," TEAA 2006, Springer 2007, in press.

[3] J. Hoffert, "Maintaining pubish/subscribe middleware QoS in dynamic envirenments," DEBS 2009, July 6-9 2009, Nashvile, TN, USA.

[4] S. Chen, P. Greenfield, "QoS evaluation of JMS : an empirical Approach," Proceedings of the 37th Hawaii International Conference on System Sciences – 2004.

[5] G. Pardo-Castellote, "OMG Data Distribution Service: Architecture Overview," IEEE 2003.

[6] Data Distribution Service for Real-time Systems Version 1.2 OMG Available Specification formal/07-01-01.

[7] M. Ryll, S. Ratchev, “Towards a publish/subscribe control architecture for precision assembly with the data distribution service,” 2008, IFIP, Volume 260, Micro-Assembly Technologies and Applications.

[8] S. Z. Mohd Hashim, “Comparative Analysis on Adaptive Features for RFID Middleware,” PICCCE 2008, in press.

[9] J. L. Poza, “Adding an Ontology to a Standardized QoS-Based MAS Middleware,” IWANN 2009, in press.

[10] J. Hoffert, “Evaluating Transport Protocols for Real-Time Event Stream Processing Middleware and Applications,” OTM 2009, in press.

[11] L C. Llewellyn, K. M. Hopkinson and S. R. Graham, “Distributed Fault-Tolerent Quality of Wireless Networks,” IEEE Transactions on Mobile Computing, Vol. 10, No. 2, February 2011.

[12] I. Chen, A. P. Speer and M. Eltoweissy, “Adaptive Fault-Tolerent QoS Control Algorithms for Maximizing System Lifetime of Query-Based Wireless Sensor Networks,” IEEE Transactions on Dependable and Secure Computing, Vol. 8, No. 2, March-April 2011.

[13] Y. Xiao, C. Lin, Y. Jiang, X. Chu and X. Shen, “Reputation-based QoS Provisioning in Cloud Computing via Dirichlet Multinomial Model,” IEE ICCC 2010.

[14] W. Abramowicz, M. Kaczmarek, D. Zyskowski, “Duality in Web Service Reliabiliy,” AICT/ICIW 2006.

[15] B. Bruhadeshwar and S. S. Kulkarni, “Balancing Revocation and Storage Trade-Offs in Secure Group Communication,” IEEE Transactions on Dependable And Secure Computing, Vol. 8, No. 1, January-February 2011.

[16] D. Choi, S. Lee, D. Won and S. Kim, “Efficent Secure Group Communications for SCADA,” IEEE Transaction on Power Delivery, Vol. 25, No. 2, April 2010.

[17] P. de Saqui-Sannes, T. Villemur, B. Fontan, S. Mota-Gonzalez, M. S. Bouassida, N. Chridi, I. Chrisment, L. Vigneron, “Formal verification of secure group communication protocols modelled in UML," Innovations in Systems and Software Engineering, Vol.6, N°1-2, pp.125-133, Mars 2010

[18] G. Muller, R. Marlet, C. Pu and A. Goel, “ Fast, Optimized Sun RPC Using Automatic Program Specialization,” Distributed Computing Systems, 18th International Conference, 26-29 May 1998, Amsterdam.

[19] Z. Cao, R. Jandhyala and S. Koduvayur, “Performa,ce Evaluation for SOAP and RFC in SAP Netweaver Platform,” Web Service (ICWS), IEEE International Conference, 5-10 July 2010, Miami, FL.

[20] P. Basanta-Val, M. Garcia-Valls and I. Estévez-Ayres, “An architecture for distributed real-time Java based on RMI and RTSJ,” ETFA, IEEE Conference, 13-16 Sept. 2010, Bilbao.

[21] A. Detti, P. Loreti, N. Blefari-Melazzi and F. Fedi, “Streaming H.264 scalable video over data distribution service in a wireless environment,” WoWMoM, IEEE International Symposium, 14-17 June 2010, Monteral, QC, Canada.