Projet IA04 - NF28

Colladia, éditeur de diagramme collaboratif

Jean Vintache
Florian Impellettieri
Charles Menier
Marouane Hammi
Adrien Jacquet

4 juin 2016

utc
Université de Technologie
Compiègne

Table des matières

1 Introduction

2 Serveur multi-agent

2.1 Description générale du système

Le serveur multi-agent a pour but de permettre la dimension collaborative de Colladia. C'est en effet son rôle d'informer les différents clients éditant un même diagramme des modifications effectuées par les autres utilisateurs. D'un point de vue technique, le serveur utilise un certain nombre de technologies :

- Les requêtes des clients sont reçues via une interface REST implémentée grâce au framework Java Restlet. Cette technologie étant par définition unidirectionnelle, les clients doivent effectuer des requêtes régulières sur le serveur pour être tenus au courant des modifications du diagramme.
- Le traitement des requêtes est effectué par un système multi-agent utilisant le framework JADE.
- Le contenu des requêtes REST et des messages du SMA sont sérialisés en JSON via la librairie Java Jackson.

La structure du SMA est divisée en deux conteneurs :

- Le conteneur principal où résident notamment les agents standards JADE (DF, AMS, etc.) ainsi que l'agent lié au serveur Restlet qui va transformer les requêtes REST reçues en messages pour le SMA (RestAgt). On y trouve aussi l'agent chargé de sauvegarder régulièrement l'état des différents diagrammes dans des fichiers de manière à pouvoir restaurer leur état après un éventuel redémarrage du serveur (SaveAgt).
- Le conteneur de diagramme contenant pour chaque diagramme :
  - Un ou plusieurs agents élément représentant le diagramme en soi ou des éléments présents dans ce dernier (EltAgt). Ces agents forment entre eux une arborescence dont l'élément représentant le diagramme est la racine.
  - Un agent horloge responsable de gérer une horloge logique pour la sauvegarde des modifications effectuées sur le diagramme (ClockAgt). L'horloge est initialisée à 0 au démarrage du serveur et est incrémentée de 1 à chaque modification du diagramme ou de ses sous-éléments.
  - Un agent historique sauvegardant une liste des dernières modifications (HistAgt).

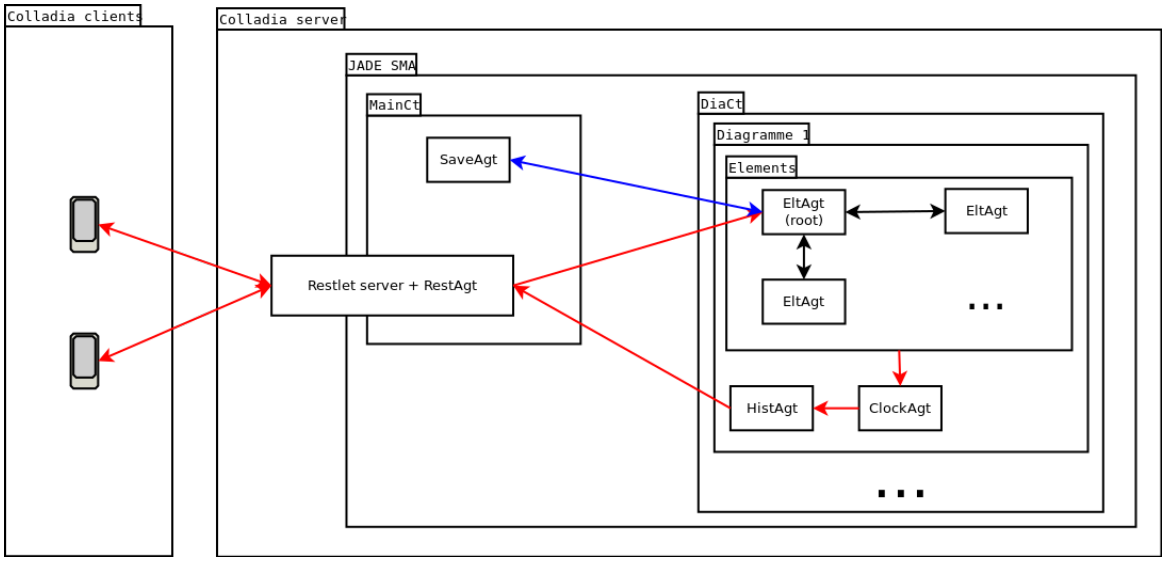

FIGURE 1 – Schéma de l'architecture générale du serveur

2.2 Description de l'interface REST

2.2.1 Champs communs

Un certain nombre de champs sont communs aux différentes requêtes REST. Pour les entrées le champ `last-clock`, optionnel, permettant au serveur de connaître la dernière valeur de l'horloge logique reçue par le client effectuant la requête.

Pour les sorties, il faut commencer par différencier les modifications sauvegardées par l'agent historique du message de retour des requêtes REST. Le retour comporte obligatoirement un champ `status` contenant un code d'erreur et un champ `clock` donnant la dernière valeur de l'horloge logique du diagramme. Le champ `status` peut prendre une des valeurs suivantes, inspirées des standards web :

- succès (2xx) :
    - 200 : OK
- redirection (3xx) :
    - 304 : non-modifié
- erreur client (4xx) :
    - 400 : requête mal formée
    - 401 : existe déjà
    - 404 : non-trouvé
- erreur serveur (5xx) :
    - 500 : erreur interne

Dans le cas où la requête n'a pas pu être exécutée (`status != 200`), un champ `error` est intégré au retour contenant une description de l'erreur. Si la requête à été exécutée correctement, qu'un champ `last-clock` a été spécifiée par le client et que sa valeur est valide par rapport aux modifications enregistrées par l'HistAgt du diagramme, alors on renvoie un champ `modification-list` qui est un tableau JSON contenant un ensemble de modifications telles que décrites ci-après. Si on est dans l'incapacité de produire une liste de modifications, on retourne un champ `description` contenant une description complète du diagramme et de ses sous-éléments sous-forme d'une sérialisation JSON.

Les modifications engendrées par certains types de requêtes (PUT, POST et DELETE) et stockées dans l'agent d'historique comportent forcément un champ `clock` indiquant l'horloge à laquelle la modification a eu lieu et un champ `type` donnant le type de la modification (PUT, POST ou DELETE).

2.2.2 Requêtes GET

Les requêtes de type GET permettent d'obtenir des informations sur l'état actuel du diagramme et n'engendrent jamais de modifications. Si l'URI est de la forme `<server adress>`, renvoie un champ `list` contenant un tableau JSON des noms des différents diagrammes stockés sur le serveur.

Si l'URI est de la forme `<server adress>/<diagram>[/<element> ...]`, renvoie un champ `path` donnant le chemin vers l'élément ciblé sous forme d'un tableau JSON ainsi qu'un champ `description` contenant la description récursive de cet élément.

2.2.3 Requêtes PUT

Les requêtes de type PUT permettent de créer de nouveaux diagrammes ou de nouveaux sous-éléments. Si l'URI est de la forme `<server adress>/<diagram>` c'est une création de diagramme, et dans le cas d'un succès (un diagramme possédant le même nom n'existe pas encore), on renvoie un champ `path` contenant le nom du diagramme.

Si l'URI cible un élément (`<server adress>/<diagram>/<element[/<element> ...]>`), on créé un nouveau sous-élément. La requête peut optionnellement contenir un champ `properties` contenant des couples (propriété, valeur) pour initialiser les propriétés de l'élément. Le retour contient un champ `path` ainsi qu'un champ `properties` contenant une description des propriétés de l'élément venant d'être créé.

2.2.4 Requêtes DELETE

Les requêtes de type DELETE permettent de supprimer un EltAgt ou bien une partie de ses propriétés. Dans les deux cas, l'URI est de la forme `<adrr>/<diagram>[/<element> ...]`. Si la requête possède un champ `properties-list` contenant une liste de nom de propriété à supprimer, alors on supprime uniquement ces propriétés au niveau de l'élément ciblé. Sinon, on supprime l'EltAgt ciblé ainsi que tous ses fils récursivement. Dans les deux cas, la modification a la même forme que la requête initiale, à ceci-près que le chemin de l'URI est intégré dans un champ `path` et le type de la requête dans un champ `type`.

2.2.5 Requêtes POST

Les requêtes REST de type POST permettent de modifier la valeur des propriétés d'un élément, une option permet d'exécuter un algorithme d'auto-positionnement sur les fils d'un EltAgt ciblé. Dans les deux cas, l'URI est de la forme `<adrr>/<diagram>[/<element> ...]`. Pour modifier directement des propriétés, un champ `properties` contenant des couples (propriété, valeur) doit être spécifié dans la requête. La modification possède une forme identique à la requête initiale.

Pour exécuter l'algorithme d'auto-positionnement, il faut spécifier dans la requête un champ `options` associé à une liste JSON contenant une valeur `auto-positioning`. Cette requête induit la sauvegarde de 0+ modifications de type POST, une par fils auto-positionnable. Un fils est dit auto-positionnable si il contient des propriétés suffisantes pour permettre un repositionnement, dans notre cas `xMin`, `xMax`, `yMin` et `yMax`. Chaque modification enregistrée contient un champ `path` contenant le chemin de l'EltAgt modifié ainsi qu'un champ `properties` récapitulant les altérations effectuées.

2.3 Description des agents et des comportements

2.3.1 RestAgt

Le RestAgt est couplé à une classe définissant un serveur Restlet de manière à pouvoir récupérer des requêtes REST puis à les transférer aux agents pouvant y répondre. L'agent est aussi responsable de la création des nouveaux diagrammes (PUT) et de fournir aux clients la liste des diagrammes existant sur le serveur (GET). À sa création, le RestAgt s'enregistre auprès du Directory Facilitator (DF) en tant qu'interface REST.

On peut dénombrer deux comportements pour cet agent. Le premier n'est pas une classe fille de la classe Behaviour, mais on peut considérer que le fait d'attendre les requêtes REST au niveau du serveur Restlet rentre dans la catégorie du comportement au sens multi-agent du terme. À la réception d'une requête, créé un nouveau diagramme ou retourne la liste des diagrammes disponibles si besoin. Sinon, envoie la requête sous forme de message ACL au EltAgt racine du diagramme ciblé par la requête et lance un ReceiveBhv. L'AID du ClockAgt du diagramme est aussi insérée au niveau du champ `reply-to` du message. Le ConversationId est généré par la classe Java UUID.

Le ReceiveBhv attend un message de retour pour une requête envoyée précédemment puis formule la réponse à la requête REST initiale du client.

2.3.2 SaveAgt

Le SaveAgt est responsable de la sauvegarde et de la restauration de l'état des diagrammes après un éventuel redémarrage du serveur. La description de chaque diagramme est stocké dans un fichier au format JSON. Le premier comportement, dit RestoreBhv, est un OneShotBehaviour lancé au démarrage de l'agent et qui, à partir des données sauvegardées, créé les différents diagrammes (EltAgt racine + ClockAgt + HistAgt) et envoie un message au nouveau EltAgt pour restaurer la valeur de ses propriétés et de ses sous-éléments. Une fois les différents messages envoyés, lance le TickerBhv et le ReceiveBhv.

Le TickerBhv est un comportement chargé d'envoyer régulièrement des messages GET aux différents EltAgt racine des diagrammes pour récupérer leur description complète. Le ReceiveBhv est un comportement cyclique qui récupère les réponses à ces messages et écrit les descriptions reçues dans des fichiers .json.

2.3.3 ClockAgt

Le ClockAgt est chargé de gérer l'horloge du diagramme et exprime un unique comportement cyclique, le ReceiveBhv. Il attend un message de type INFORM provenant d'un EltAgt, incrémente l'horloge si le message ne comporte par d'option `no-history` et est de type POST, PUT ou DELETE, puis, dans tous les cas, ajoute un champ `clock` au message comportant la valeur la plus récente de l'horloge. Le message est ensuite envoyé au HistAgt.

2.3.4 HistAgt

Le HistAgt est l'agent chargé de gérer l'historique des modifications et comporte un unique CyclicBehaviour encore une fois nommé ReceiveBhv. Il réceptionne les messages envoyés par le ClockAgt, et si le message ne comporte par d'option `no-history` et est de type POST, PUT ou DELETE, ajoute le contenu du message, la modification, à l'historique.

L'HistAgt est aussi chargé de formuler la réponse à la requête REST initiale si le message reçu du ClockAgt ne comporte pas une option `no-reply`. Dans ce cas, si le message intègre un champ `last-clock` et que sa valeur est supérieure à l'horloge de la première modification encore stockée dans la liste des modifications, l'agent renvoie au RestAgt la liste des modifications appliquée depuis celle portant l'horloge `last-clock`. Sinon envoie un message au EltAgt racine pour récupérer la description complète du diagramme en ajoutant un champ `reply-to` vers le RestAgt.

2.3.5 EltAgt

Agent chargé de stocker l'état courant d'un élément ou du diagramme lui-même. Pour chaque diagramme, les EltAgt forment une arborescence et la racine, qui représente le diagramme en soi, s'enregistre auprès du DF. Chaque EltAgt implémente des fonctions permettant entre autre d'ajouter, supprimer, modifier des éléments. C'est aussi eux qui délibèrent pour permettre un auto-positionnement des éléments à la demande du client.

2.3.5.1 ReceiveBhv

Le principal comportement utilisé, le ReceiveBhv, est un CyclicBehaviour lancé au démarrage de l'agent et qui attend des messages de requêtes des autres agents, que ce soit du RestAgt, du SaveAgt ou d'un autre EltAgt du diagramme. Le fonctionnement de ce comportement après réception d'un message de type REQUEST peut être résumé comme ceci :

- PUT :
    - Si l'élément courant est l'avant-dernier élément de l'URI, ajoute un nouvel élément si celui-ci n'existe pas déjà (création d'un EltAgt et ajout à l'arborescence).
    - Sinon, transfert du message au fils correspondant au prochain élément dans l'URI si il existe ou retourne une erreur.
- GET :
    - Si l'élément courant a une profondeur dans l'arborescence strictement inférieure à celle de la cible de l'URI, transfert du message au fils correspondant au prochain élément dans l'URI si il existe ou retourne une erreur.
    - Sinon :
        - Si l'élément est une feuille, répond au message en envoyant la liste des propriétés et leurs valeurs.
        - Sinon, transfert le message à tous les sous-éléments et lance un comportement WaitDescriptionBhv pour attendre les réponses.

- DELETE :
  - Si la requête contient un champ `properties-list` (suppression de propriété) :
    - Si l'élément courant est le dernier élément de l'URI, suppression des propriétés dans l'élément si elles existent.
    - Sinon, transfert du message au fils correspondant au prochain élément dans l'URI si il existe ou retourne une erreur.
  - Sinon (suppression d'élément) :
    - Si l'élément courant a une profondeur dans l'arborescence strictement inférieure à celle de la cible de l'URI, transfert du message au fils correspondant au prochain élément dans l'URI si il existe ou retourne une erreur.
    - Sinon :
      - Si l'élément courant est le dernier élément de l'URI, envoie une réponse à l'expéditeur initiale de la requête.
      - Transfert du message à tous les fils.
      - Autodestruction de l'agent.
- POST :
  - Si l'élément courant est le dernier élément de l'URI :
    - Si le message comporte une option `auto-positioning` (auto-positionnement des fils) :
      - Envoi un message de type CFP (Call For Proposal) à tous les fils comportant un angle pour la dispersion ainsi que la largeur et la longueur de l'élément courant pouvant prendre la valeur -1 pour indiquer une taille infinie.
      - Lance un nouveau comportement WaitProposalBhv avec en paramètre le nombre de messages envoyés vers les fils.
    - Sinon, si le message comporte un champ `properties`, modification des propriétés de l'élément et envoi une réponse à l'expéditeur initial de la requête.
  - Sinon, transfert du message au fils correspondant au prochain élément dans l'URI si il existe ou retourne une erreur.
- RESTORE (type supplémentaire du serveur pour la restauration des diagrammes) :
  - Pour chaque couple (clef, valeur) du champ `properties` :
    - Si la valeur est un objet JSON, créé un nouvel élément, l'ajoute à l'arborescence et lui envoie un message de restauration contenant la valeur du couple.
    - Sinon, c'est une propriété, et on l'ajoute à celles de l'élément courant.

2.3.5.2 Autres comportements

Le comportement WaitDescriptionBhv est lancé après avoir envoyé des messages aux fils suite à un GET et attend leurs descriptions respectives. Une fois la description de tous les fils reçus, formule une réponse au message de requête reçu initialement (en prenant en compte le `reply-to`).

Le ReceiveCFPBhv est un comportement cyclique lancé au démarrage de l'agent qui, à chaque réception d'un message CFP par l'agent lance un nouveau ProposeBhv. Ce dernier commence par placer l'élément au centre de son père, puis envoi ses nouvelles coordonnées (`x` et `y`) ainsi que ses

dimensions (`width` et `height`) au père dans un message PROPOSE. Tant que ce dernier renvoi des REJECT_PROPOSAL, avance d'un pas fixé dans la direction donnée par l'élément père au lancement de l'algorithme d'auto-positionnement. Une fois un ACCEPT_PROPOSAL reçu, envoi un message au HistAgt du diagramme pour enregistrer la modification des coordonnées. Sachant que plusieurs modifications peuvent être engendrées par une seule requête d'auto-positionnement et qu'une requête REST n'accepte qu'une unique réponse, on ajoute un flag `no-reply` au message, de manière à ce que ce message n'engendre pas une réponse au RestAgt qui serait ensuite transférée au client. Si, lors d'une itération l'élément ne peut plus se déplacer ni en x ni en y car il a atteint les limites de son père, alors il intègre une option `force` à son message pour forcer l'acceptation de la proposition. Si l'élément ne comporte pas les propriétés nécessaires aux calculs d'auto-positionnement (`xMin`, `xMax`,`yMin` et `yMax`), il renvoi un message ayant un performatif REFUSE.

Le comportement WaitProposalBhv est lancé au début de chaque algorithme d'auto-positionnement en sachant combien de CFP ont été envoyés à des fils et entretien une liste des coordonnées et dimensions des éléments fils dont la nouvelle position a déjà été acceptée. À la réception d'un message de type REFUSE, décrémente le compteur de message attendu. À la réception d'un message PROPOSE, si la nouvelle position de l'élément fils chevauche celle d'un des éléments déjà accepté, retourne un message de type REJECT_PROPOSAL, sinon, un ACCEPT_PROPOSAL. Dans le cas où le message reçu comporte une option `force`, renvoie toujours un ACCEPT_PROPOSAL. Chaque envoi d'un ACCEPT_PROPOSAL induit une décrémentation du compteur. Une fois que le compteur atteint 0, répond au message de requête initial en ajoutant une option `no-history`, de manière à fournir une réponse à la requête REST incluant uniquement les modifications réalisées par ses fils.

2.4 Description générale des messages

2.5 Limites et améliorations

3 Interface client