PROJET RNTL ARCAD*

D3.3 - Démonstration de test mettant en évidence différents protocoles d'interaction

T0+21 : document public

David Emsellem, Mireille Blay, Audrey Occello

Laboratoire I3S - Bâtiment ESSI
BP 145
06903 Sophia Antipolis CEDEX

15 septembre 2002

Résumé :
Ce document présente au travers d'un exemple, l'utilisation du service d'interaction ainsi que différents modèles d'interaction. La lecture du livrable D3-2 est nécessaire à la compréhension de ce document. L'ensemble du code source du service d'interaction et de l'application de démonstration est accessible à l'adresse http://noah.essi.fr *sous une licence GNU Lesser General Public License (LGPL :* http://www.gnu.org/licenses/lgpl.html).

1. Description de l'application

L'application choisie concerne la gestion d'un ensemble d'agenda personnel. Elle est essentiellement composée de trois composants Java/RMI complètement indépendants :

- Un composant Agenda permettant de stocker des rendez-vous. Ce composant sera instancié quatre fois afin de construire l'agenda de l'équipe Rainbow et les agendas personnels de David, Michel et Mireille.
- Un composant SecurityService permettant de valider des appels de méthodes. Ce composant sera instancié une seule fois pour constituer le service de sécurité.
- Un composant Display servant de console pour l'affichage de message. Ce composant sera instancier trois fois afin de permettre l'affichage des agendas de l'équipe, et de michel sous deux configurations différentes.

---

* Le projet ARCAD (Architecture Extensible pour Composants Adaptables) a été labellisé en décembre 2000. Les différents partenaires sont France Télécom R\&D (équipe ASR - Thierry.Coupaye@francetelecom.com), INRIA (projet Oasis - Denis.Caromel@inria.fr, projet Sardes - Daniel.Hagimont@inria.fr), Ecole des Mines de Nantes (equipe OCM - Thomas.Ledoux@emn.fr), le laboratoire I3S commun à l'Université de Nice - Sophia Antipolis et au CNRS (projet Rainbow - Anne-Marie.Pinna@unice.fr). Le coordonateur du projet est Michel.Riveill@unice.fr}

A l'aide des interactions logicielles (voir le livrable D2-3) nous souhaitons lier dynamiquement les différentes instances de ces composants afin d'obtenir le graphe d'interaction suivant présenté sur la *figure 1*.

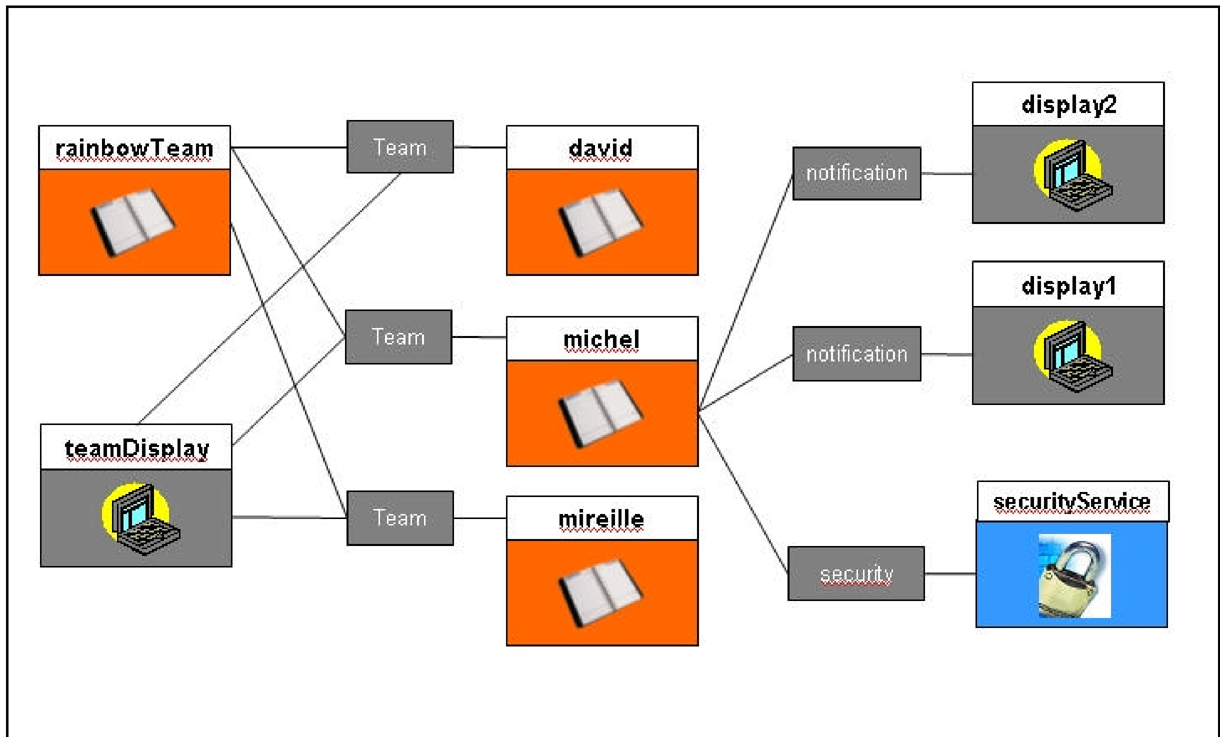

Figure 1 - Graphe des interactions de l'application agenda

2. Préparation des composants

Afin que les futures instances des composants puissent recevoir des interactions (être liées et déliées dynamiquement), le code de leur classe doit être modifié afin de pouvoir intercepter les messages reçus par le composant.

En Java/Java RMI, cette tâche est réalisée par un outil de modification de byte code nommé **GenInt** (*noah.tools.GenInt*) qui utilise la bibliothèque d'instrumentation de class file java BCEL [BCEL].

Le code ajouté doit permettre au composant de:

- Stocker les différentes règles d'interactions pour une méthode donnée
- Fusionner/ Défusionner dynamiquement les règles d'interaction
- Donner la main à un contrôle local lors de la réception du message déclencheur
- S'enregistrer auprès d'un serveur d'interaction sous un nom donné

L'outil d'instrumentation **GenInt** effectue les modifications suivantes:

- Pour chaque méthode métier (implémentation d'une méthode d'interface), la méthode est renommée (en la préfixant) et une méthode portant la signature d'origine est redéfinie. Celle-ci peut alors intercepter les messages reçus par le

composant. Le code de cette méthode consiste à exécuter la règle d'interaction associée à la méthode s'il y a lieu où à exécuter le code initial dans le cas inverse.

- Plusieurs méthodes permettant la prise en compte des interactions par le composant sont ajoutées:
    - `int addRule(String methodSignature, RuleInstance)`
      Ajout d'une règle pour une méthode donnée (fusion implicite)
    - `Object invokeMethod(String methodSignature, Object[] params)`
      Invocation dynamique d'une méthode sur le composant
    - `void removeRule(String methodSignature, Object[] params)`
      Retrait d'une règle pour une méthode donnée (fusion implicite des règles restantes)
    - `String getBehaviorOn(String methodSignature)`
      Pour obtenir la règle associée à une méthode donnée (sous forme ISL)
    - `String getLogicalName()`
      Pour obtenir le nom logique de l'objet (le nom sur le serveur d'interaction)
    - `void setLogicalName(String logicalName)`
      Pour enregistrer l'objet auprès du serveur d'interaction
- Création d'un champ statique contenant une méta-classe (de type noah.vm.MetaClass) permettant la pose de règle sur les méthode statiques et constructeurs du composant. La méta-classe permet aussi d'obtenir le slot d'une méthode dans la table de méthode associée à l'objet.
- Création d'un champ contenant la table de méthode (de type noah.vm.MethodTable). Celle-ci permet le binding méthode/règle pour chaque méthode d'interface.
- Instrumentation du code des constructeurs afin d'initialiser la table de méthodes en fonction de la méta-classe.

Dans notre exemple 3 classes doivent être modifiée via l'outil **GenInt**. L'outil est invoqué en lançant la classe *noah.tools.GenInt* comme suit:

Lignes de commande:

```
> java noah.tools.GenInt -type rmi noah.samples.agendas.AgendaImpl

> java noah.tools.GenInt -type rmi noah.samples.agendas.SecurityServiceImpl

> java noah.tools.GenInt -type rmi noah.samples.agendas.DisplayImpl
```

3. Lancement du serveur d'interaction

Une fois l'application lancée (i.e. ses différents composants crées), nous devons lancer un serveur d'interaction. Celui-ci peut bien sûr avoir été lancé préalablement dans le contexte d'une autre application. Dans notre exemple nous utiliserons la version RMI du serveur d'interaction.

Pour lancer un serveur d'interaction rmi, il suffit d'exécuter la commande suivante:

```
> java noah.server.rmi.IslServerImpl
```

4. Création et nommage des composants

L'application consiste en un ensemble d'objets distribués RMI complètement indépendant :

- 3 agendas personnels et 1 agenda d'équipe
- 3 afficheurs
- 1 service de sécurité (service joué par une boite de dialogue Oui/Non)

Nous décrivons ici comment créer puis nommer un composant en prenant comme exemple l'authentificateur (composant SecurityServiceImpl) qui sera nommé securityService. Le code présenté ici correspond au code java à insérer dans la classe souhaitant créer le composant.

1) On commence par créer une instance du composant de manière standard (i.e. comme en Java) :

```
SecurityService securityService=new SecurityServiceImpl();
```

2) On enregistre ensuite ce composant dans un registre RMI afin qu'il soit disponible à distance (la aussi comme en Java RMI standard) :

```
import javax.naming.InitialContext;

...

try {
      System.out.println("JNDI object registration");
      InitialContext context=new InitialContext();;
      context.rebind("rmi_security",securityService);
} catch (Exception e){
      System.err.println("error: "+e.getMessage());
}
```

3) Enfin on enregistre le composant auprès du serveur d'interaction (action spécifique pour que le composant ait une existence auprès du service d'interaction). On parle de *nom logique* lorsqu'il s'agit d'une référence Noah.

```
import noah.common.NamingService;

...

try {
	System.out.println("NOAH object registration");
	NamingService.bind("SecurityService",securityService);
} catch (Exception e){
	System.err.println("error: "+e.getMessage());
}
```

Nous procédons de même pour les autres composants de l'application.

- 3 afficheurs (composant ***Display***)
  - A afficheur d'équipe nommé **TeamDisplay** (nom logique)
  - Un nommé **Display2**
  - Un nommé **Display1**

- 4 agendas (composant ***Agenda***)
  - Un agenda d'équipe nommé **RainbowTeam**
  - Un afficheur pour Michel nommé **RiveillMichel**
  - Un afficheur pour David nommé **EmsellemDavid**
  - Un afficheur pour Mireille nommé **BlayMireille**

NB: De manière transparente la méthode *bind(String name, Object component)* de la classe NamingService appellera la méthode *setLogicalName(String name)* du composant. La méthode *setLogicalName* n'existant pas lors de la compilation mais uniquement après modification par l'outil GenInt, la méthode *bind* effectue un safe-cast afin d'éviter une erreur de compilation.

5. Enregistrement des schémas d'interaction

Avant de pouvoir liés les composants de l'application à l'aide des interactions, il est nécessaire d'écrire les schémas d'interaction puis de les enregistrer auprès du serveur d'interaction. Dans l'application de démonstration, nous utilisons 3 schémas d'interactions.

5.1 Le schéma notification

L'interaction suivante permet pour chaque appel reçu sur l'objet **obj** auquel est attaché l'interaction d'exécuter, l'appel initialement prévu (**obj**._call) puis d'appeler la méthode notify de l'objet **display** qui reçoit en paramètre l'appel initial (_call).

```
interaction notification(Object obj, Display display){
	obj.* -> obj._call ; display.notify(_call)
}
```

*code ISL du schema **notification***

5.2 Le schéma security

L'interaction suivante permet pour chaque appel reçu sur l'objet **obj** auquel est attaché l'interaction d'exécuter, d'appeler un authentificateur **service** et selon sa réponse d'effectuer l'appel initial ou de lever une exception

```
interaction security(Object obj, SecurityService service){
   obj.* -> if service.check(_call) then
                              obj._call
                        else
                              exception "OperationRefused"
                        endif
}
```

code ISL du schema ***security***

5.3 Le schéma team

L'interaction suivante permet lors d'un ajout de rendez-vous dans un agenda d'équipe **team** d'ajouter également le rendez-vous dans l'agenda d'un membre **member** (première règle d'interaction). L'interaction entraîne également l'affichage d'un message dans un afficheur **teamDisplay** (dédié à l'équipe) lorsque que le rendez-vous est ajouté dans l'agenda du membre (2ème règle d'interaction). Cette interaction met en évidence la propagation suivante:

team.addMeeting(m) ? **member**.addMeeting(m) ? **teamDisplay**.notify(*_call*)

```
interaction team(Agenda team, Agenda member, Display teamDisplay){
   team.addMeeting(Meeting meeting)
      ->  team._call;
          member.addMeeting(meeting),

   member.addMeeting(Meeting meeting)
      -> member._call;
          teamDisplay.notify(_call)
}
```

code ISL du schéma ***team***

5.4 Enregistrement des schémas d'intéraction

Une fois les différents schémas d'interaction décrit, il est nécessaire de les enregistrer dans le serveur d'interaction. Nous donnons ci-dessous le code java d'une méthode permettant à partir d'un fichier texte **filename** contenant un schéma d'interaction sous une forme ISL d'enregistrer le schéma auprès d'un serveur d'interaction.

```java
import java.io.BufferedReader;
import java.io.FileReader;
import noah.common.NoahHome;
import noah.server.IslServer;

...

void registerPattern(String fileName) throws Exception {
   String islText="";
   // reading ISL pattern description from file
   try {
         BufferedReader br = new BufferedReader(new
         FileReader(fileName));
         while (br.ready()) {
           islText += br.readLine();
           islText += "\n";
         }
         br.close();
   } catch (Exception e) {
         System.err.println("error: "+e.getMessage());
         return;
   }

   // now send islText to noah server. It will be parsed and
   // store on the server
   try {
         IslServer noahServer=NoahHome.getServer();
         String pname=noahServer.createPattern(islText);
         System.err.println("pattern "+pname+" registered.");
   } catch (IslsException i1) {
         System.err.println("error: can't register pattern!");
   }

}
```

En supposant que nos 3 schémas d'interactions soient contenus dans 3 fichiers respectivement nommés **security.isl, notification.isl et team.isl.** L'enregistrement par le code java suivant

```java
registerPattern("security.isl");
registerPattern("notification.isl");
registerPattern("team.isl");
```

7. Instanciation des schémas

Une fois les schémas enregistrés dans le serveur d'interaction, il est possible de les utiliser en les instanciant. L'instanciation est réalisée par le serveur d'interaction en lui adressant une requête contenant le nom d'un schéma d'interaction ainsi que les noms logiques des instances participant à l'interaction. Le serveur unifiera ces paramètres avec les paramètres formels du schéma d'interaction.

Supposons que l'on veuille instancier le schéma **security** (c.f. 6.4) entre l'agenda michel nommé **RiveillMichel** (c.f. 4) et l'authentificateur **SecurityService**. Le code Java à produire est le suivant:

```
import noah.server.util.Instanciator;
import noah.server.IslServer;
import noah.common.NoahHome;

...

ISLServer noahServer=NoahHome.getServer();
try {

     Instanciator instanciator=new Instanciator("security");
     instanciator.addParameter("RiveillMichel");
     instanciator.addParameter("SecurityService");
     instanciator.execute(noahServer);

} catch (Exception e) {
     e.printStackTrace();
}
```

Dans notre application, le graphe d'interaction de la *figure 1* est obtenu en instanciant chaque schémas comme suit:

Code java d'instanciation:

```
public createInteractions(){
     instanciate("security",
                new String[]{"RiveillMichel", "SecurityService"}
     );
     instanciate("notification",
                new String[]{"RiveillMichel", "Display1"}
     );
     instanciate("notification",
                new String[]{"RiveillMichel", "Display2"}
     );
     instanciate("team",
      new String[]{"RainbowTeam","EmsellemDavid","TeamDisplay"}
     );
     instanciate("team",
       new String[]{"RainbowTeam","BlayMireille","TeamDisplay"}
     );
     instanciate("team",
      new String[]{"RainbowTeam","RiveillMichel","TeamDisplay"}
     );
 }


public instanciate (String patternName, String[] names){
     ISLServer noahServer=NoahHome.getServer();
     System.out.println("instanciate "+patternName);
     try {
     Instanciator instanciator=new Instanciator(patternName);
      for (int i=0;i<names.length;i++){
     instanciator.addParameter(names[i]);
     }
     instanciator.execute(noahServer);
     } catch (Exception e) {
     e.printStackTrace();
     }
```

}

NB: un outil (voir *figure 2*) permet d'effectuer les tâche précédemment décrite et ce de manière graphique. L'outil permet d'éditer et d'enregistrer des schémas d'interaction, d'instancier des schémas, de détruire certaines interactions, mais aussi de visualiser l'état d'un composant (les règles d'interactions qui lui sont associées). L'outil peut être lancé en ligne de commande comme suit:

> java **noah.gui.editor.NoahEditor**

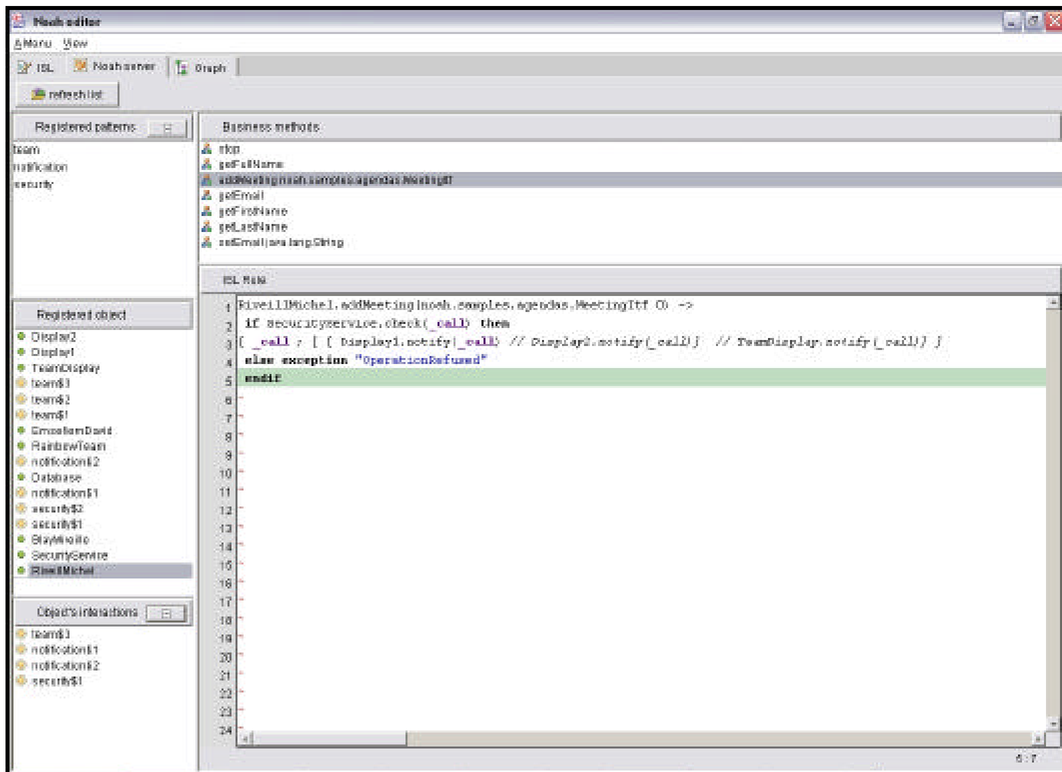

figure 2 – **NoahEditor**: outil d'administration du serveur d'interactions

10. Conclusion

Nous avons vu dans ce document quelles étaient les étapes nécessaires à la construction d'une application au moyen des interactions. L'exemple que nous avons choisi de présenter ne présente les interactions au travers d'un exemple à base de composants Java RMI. Il est également possible de les utiliser avec des objets Java pur mais également avec des composants EJB, en effet le service d'interaction a été ajouté à la plate-forme EJB Jonas distribuée par le consortium ObjectWeb. Une couche d'abstraction du service d'interaction permet de lier dynamiquement des composants de nature différentes (i.e. des composants Java, Java RMI et Enterprise Java Beans).

Dans ce document nous n'avons pas décrit le fonctionnement de l'algorithme de fusion des règles d'interaction utilisé chaque fois que 2 règles sont déposées sur le même objet. Cet algorithme respecte des propriétés strictes de commutativité et d'associativité et sont

décrites dans le livrable (D3-1). L'architecture complète du service d'interaction est elle décrite dans le livrable (D3-2).

11. Références

[BCEL] *Byte Code Engineering Library,* http://bcel.sourceforge.net
[D3-1] *Spécification d'un modèle d'interaction,* http://arcad.essi.fr/documents-public/d3-1.pdf
[D3-2] *Spécification de mise en oeuvre,* http://arcad.essi.fr/documents-public/d3-2.pdf
[Jonas] *JavaTM Open Application Server*, http://www.objectweb.org/jonas/index.html
[Noah] Site web permettant le téléchargement de *Noah,* http://noah.essi.fr