A l'attention de
François Gatto
(Maître de stage et enseignant-tuteur)

Mathieu Soum
L3 Informatique 2012 - 2013
TER réalisé à l'IRIT
Du 15 avril 2013 au 7 juin 2013

Rapport de TER

Développement d'un outil de communication avec le logiciel de simulation TRNSys

Irit, Université Paul Sabatier,
118 Route de Narbonne,
31400 Toulouse

A l'attention de
François Gatto
(Maître de stage et enseignant-tuteur)

Mathieu Soum
L3 Informatique 2012 - 2013
TER réalisé à l'IRIT
Du 15 avril 2013 au 7 juin 2013

Rapport de TER

Développement d'un outil de communication avec le logiciel de simulation TRNSys

Irit, Université Paul Sabatier,
118 Route de Narbonne,
31400 Toulouse

Je souhaite adresser mes remerciements à tous les professeurs et intervenants que j'ai rencontré durant cette année universitaire.
Je souhaite remercier spécialement François Gatto pour m'avoir proposé ce TER bien évidemment mais aussi pour m'avoir épaulé et soutenu durant toute cette expérience. Il a su me communniquer son enthousiasme et sa passion dans ce qu'il fait. Il m'a fait me sentir utile à son travail de recherche et pour cela je l'en remercie.
Enfin, je tiens également à remercier Ophélie Fraisier, Antoine de Roquemaurel et Clément Vannier simplement pour avoir été là et m'avoir insufflé ce qu'il me manquait de motivation et d'énergie dans les moments de fatigue.

Table des matières

Chapitre 1

Introduction

Pour cette expérience de fin de licence, je me suis tourné vers un TER plutôt que vers un stage pour plusieurs raisons.

La première est que j'avais déjà songé à faire un stage à l'IRIT l'année dernière à la fin de mon DUT. J'avais envie de découvrir le milieu de la recherche et de voir comment fonctionnait ce milieu dont on ne nous parle que très peu lorsqu'on nous parle d'orientation. Je m'étais cependant dirigé vers une autre proposition dont le sujet m'interessait particulièrement et qui me donnait la possibilité de poursuivre ma mission durant le mois de Juillet.
La deuxième raison est que je n'ai pas trouvé de sujet de stage qui me corresponde et m'intéresse suffisamment pour me donner envie de m'investir pendant 2 mois. Lorsque que les stages sont courts comme celui-ci, les entreprises n'offrent que des missions sur des projets sans réel potentiel d'avenir. Ce qui peut évidemment se comprendre vu que, pour la plupart, nous ne restons pas dans l'entreprise après la période de stage. Or, le sujet de TER que m'a propsé François a un avenir et sera réutilisé par l'équipe pour qui j'ai travaillé. Il me donne la possibilité de contribuer à un projet concret et d'aider à le faire progresser.

Mon maître de stage, François Gatto fait parti de l'équipe SMAC [1]. Son projet actuel est de créer un système multi-agent qui pourra réguler l'énergie consommée dans un bâtiment tout en gardant des constantes de confort et de bien-être pour les occupants. Pour faire fonctionner ce système lors de tests à grande échelle, les différents bâtiments sont modélisés par un logiciel de simulation appellé TRNSys. Ce logiciel permet de créer ses propres composants qui seront liés à des capteurs virtuels fournissant des données en fonction des caractéristiques du bâtiment modélisé, de la période de la journée, du jour de l'année, etc. C'est ici que mon travail commence. Mon objectif est de créer une application qui communiquera avec un composant intégré à TRNSys, représenté sous forme de bibliothèque dynamique. Ce composant représente d'une part le système multi-agent pour le logiciel de simulation et d'autre part, sert à réellement communiquer les données de simulation au système multi-agents.

Ce sujet m'a tout de suite intéressé car j'ai déjà travaillé sur un projet concernant le bâtiment et plus précisemment le génie climatique hors du cadre universitaire. Ayant apprécié cette expérience et les connaissances qu'elle m'a apporté, j'ai souhaité approfondir la découverte de ce domaine en postulant pour ce TER.
J'ai organisé la synthèse de cette nouvelle expérience en trois parties majeures.

- Premièrement, la description du contexte dans lequel s'est déroulé ce TER.
- Deuxièmement, la partie plus technique sur le travail réalisé pendant cette période.
- Troisièmement, enfin, un bilan personnel et professionnel de cette expérience.

Pour une meilleure compréhension de ce rapport, un glossaire a été dressé dans les annexes. La plupars des termes techniques y sont définis.

Je précise ici quelques indications typographiques sur ce document. Les élément écrit en utilisant une police à `chasse fixe` sont des élément relatifs au code source produit. Ce sont des noms de paquetage, de classe, de méthode ou encore de variables. Les acronymes sont détaillés en note de pied de page. Pour les acronymes en anglais, leur traduction est donnée *en italique*.

1. **S**ystèmes **M**ulti-**A**gents **C**oopértifs

Chapitre 2

Contexte de travail

2.1 Présentation de l'IRIT

J'ai effectué mon stage au sein de l'IRIT, une unité mixte de recherche comprenant 19 équipes de recherche réparties selon sept thèmes :

- Analyse et synthèse de l'information
- Indexation et recherche d'informations
- Interaction, Coopération, Adaptation paR l'Expérience (ICARE)
- Raisonnement et décision
- Modélisation, algorithmes et calcul haute performance
- Architecture, systèmes et réseaux
- Sûreté de développement du logiciel

J'ai pour ma part intégré à l'équipe SMAC dépendant du thème ICARE.

L'équipe SMAC

Créée en 1994, l'équipe Systèmes Multi-Agents Coopératifs est aujourd'hui le résultat d'un processus auto-organisationnel de chercheurs convergeant de plusieurs horizons : intelligence artificielle distribuée, systèmes distribués, simulations sociales, optimisation par recherche locale. Les travaux de l'équipe portent sur la conception de systèmes complexes et plus particulièrement de systèmes auto-adaptatifs à fonctionnalité émergente.

Aujourd'hui confirmée par les faits, la problématique scientifique de l'équipe SMAC s'inscrit dans une évolution de l'étude des systèmes naturels et artificiels selon trois dimensions :

Diversité leur normalisation ne peut faire face à l'hétérogénéité et la miniaturisation favorise la distribution en les rendant pervasifs et enfouis.

Complexité malgré l'apport des méthodes de conception, la puissance des ordinateurs et leur interconnexion en réseau accroissent sans cesse la complexité des applications.

Dynamique les possibilités d'échange avec des environnements évolutifs, dont les variations sont peu ou mal connues lors de la conception, conduisent à une spécification nécessairement incomplète.

La recherche porte sur la conception de systèmes informatiques et l'analyse d'organisations sociales robustes et pérennes évoluant de façon autonome pour s'adapter aux évolutions de l'environnement. Les adaptations devront se faire durant l'activité pour tendre en permanence vers un fonctionnement plus adéquat, ce qui justifie le concept d'auto-adaptation. Dans un tel contexte, il est illusoire de penser que l'humain puisse contrôler et piloter à distance le système ou les entités qui le composent. Au contraire, l'autonomie du système ou de ses composants permettra à celui-ci de s'administrer, s'organiser et s'adapter. Ainsi, l'auto-adaptation est elle au coeur de ces travaux.

2.2 Organisation spacio-temporelle

N'étant pas en stage mais en TER, j'ai travaillé depuis chez moi en faisant des comtpes-rendus à mon maître de stage afin de le tenir au courant de l'avancement de la mission. Nous avons fait une réunion avant de démarrer le TER afin de présenter le sujet en détail et définir les objectifs à atteindre. J'ai également effectué quelques visites supplémentaires à l'IRIT afin d'obtenir des précisions sur certains points.

Les comptes-rendus ont été fait à une fréquence moyenne de 3 par semaine. Ce qui permettait de repecter une emploi du temps tout en gardant une flexibilité si le travail n'était pas terminer à temps pour des raisons pratiques mais aussi théoriques.

2.3 Outils utilisés

2.3.1 Matériel

Le matériel utilisé pendant ce stage m'appartient – du moins pour le développement et les tests en local. J'ai donc utilisé un ordinateur fixe en tant que station de développement et un ordinateur portable en tant que station de test pour les communications réseau. Ces équipements sont reliés via le routeur de mon FAI [1] sur le réseau local de mon appartement.

2.3.2 Logiciel

Chacune de mes machines fonctionne sous Debian 7 (Wheezy). L'architecture des processeurs est 64 bits. Pour l'écriture de code, j'ai utilisé l'EDI [2] Eclipse 4.2.2 (Juno). Pour la compilation du Java, j'ai utilisé l'OpenJDK 7 disponible sur les dépôts officiel de ma distribution Debian.

Les tests du système multi-agent sont effectués grâce à un logiciel de simulation apellé TRNSys. Ce logiciel ne fonctionnant qu'en environnement Windows, j'ai donc mis en place une machine virtuelle avec Windows XP en tant que système d'exploitation et installé TRNSys sur cette machine – TRNSys m'étant fourni par mon maître de stage. J'y ai aussi installé l'Oracle JRE 7 pour exécuter la machine virtuelle Java et Microsoft Visual Studio 2010 pour développer la bibliothèque dynamique en C++. Les logiciels développés par Microsoft à licence propriétaire cités ici ont été récupérés sur la plateforme Dream Spark grâce à l'accès fournit par l'université.

Nous avons utilisé un logiciel de versionnement pour partager notre code source et garder une traces des précédentes versions. Nous nous sommes tournés vers Git et la plateforme en ligne GitHub qui fournit ce service gratuitement et également des statistiques sur le code produit.

François m'a fourni l'acces à un wiki qui nous a été d'une grande utilité. Cet outil a été au centre du TER durant toute la période. Par exemple, il nous a permis de garder contact, tenir un planning, partager des documents comme de la documentation ou encore avoir une vue globale du travail fait et du travail restant.

Le wiki contient également plusieurs schémas explicatifs illustrant ou résumant une partie du texte auquel ils étaient rattachés. La grande partie de ces schémas a été faite sur le logiciel UMLet. Ce logiciel permet de faire tout un tas de diagrammes différents, notamment ceux de la norme UML, rapidemment et de qualité.

1. **F**ournisseur d'**A**ccès **I**nternet
2. **E**nvironnement de **D**éveloppement **I**ntégré

Chapitre 3

Analyse et réalisation

3.1 Problématique et objectifs

3.1.1 Problématique

François travaille actuellement sur un système multi-agent capable de réguler l'énergie consommée dans un bâtiment pour le chauffage ou pour la climatisation tout en gardant un certains confort pour les habitants correspondant à une valeur étalon définie. Les données ambiantes sont récupérées grâce à des équipements de domotique et des capteurs présents dans le bâtiment. Cependant, pour faire des tests et vérifier que son travail fonctionne, il a besoin de tester sur une grand nombre de bâtiments avec des structures différentes.
Or, cette solution n'est physiquement pas envisageable pour des raisons évidentes. C'est pourquoi TRNSys a été choisi pour faire ces tests.

TRNSys est un logiciel capable de simuler le fonctionnement d'équipements présents dans un bâtiment en fonction de l'orientation, de la taille des pièces, de l'agencement, de l'isolation, du jour, de l'heure, du nombre d'occupants, du type de bâtiment (usine, maison individuelle, etc.) et bien d'autres critères. De plus, ce logiciel permet de créer ses propres composants à ajouter dans les modèles de bâtiment. Ainsi, en créant un composant qui peut communiquer avec le système multi-agent, ce dernier peut récupérer des données relative au bâtiment au cours de la simulation, faire les calculs relatifs à ces données et les retourner au composant pour être réintégrées dans la simulation du bâtiment.

La problématique de ce TER est donc que cette communication puisse se faire dans des contraintes de temps permettant de lancer un grande série de simulations. À mon arrivée, la communication se faisait déjà mais seulement via des objets sérialisés et prenait trop de temps. D'un point de vue plus technique, un composant TRNSys est une bibliothèque dynamique en C++ ou Fortran comprenant un seul point d'entrée : une méthode prenant en paramètre les données d'entrée du composant, les données de sortie en mise à jour et des paramètres sur l'état actuel de la situation. Cette bibliothèque appelle du code Java via la bibliothèque JNI [1] d'où le transfert initial d'objets sérialisés. Il faut donc s'assurer que l'interface C++/Java fonctionne correctement.

3.1.2 Objectifs

L'objectif premier de ce TER est de réduire la latence réseau exisante lors de l'échange d'information ente TRNSys et l'application Java. Actuellement, la latence réseau est d'environ 250 millisecondes pour un échange de 300-400 octets sur un réseau local. L'objectif à atteindre serait de descendre dans des valeurs de l'ordre de 10-30 millisecondes.
Ensuite, il faudra implémenter d'autres types de communication puis comparer les différents temps de transfert de ces derniers. Les moyens de communication en question sont :

- Objets sérialisés
- XML [2]
- JSON [3]
- Texte brut

Enfin, s'il reste du temps, il faudrait implémenter une communication via le bus logiciel Ivy [4]. Aussi, il faudrait que l'application puisse communiquer en appliquant le standard oBIX [5]

1. **J**ava **N**ative **I**nterface *Interface native Java*
2. e**X**tended **M**arkup **L**anguage
3. **J**ava**S**cript **O**bject **N**otation
4. Une fiche technique est disponible en annexe
5. **o**bject **B**uilding **I**nformation e**X**change. Une fiche technique descriptive est disponible en annexe

3.2 Analyse et solution choisie

3.2.1 Étude de la latence réseau

Cette latence réseau est calculée pour chaque pas de la simulation. J'ai donc décidé de calculer le temps de chaque appel de méthode fait par le code Java durant un pas de simulation. En isolant les différentes méthodes, je pourrai isoler la partie du code qui prend "trop" de temps et essayer de l'optimiser. J'effectuerai également plusieurs tests selon des configurations différentes. Avec le matériel à ma disposition, je peux tester des échanges sur un réseau local câblé ou via Wi-Fi en plus des tests entre la machine virtuelle et la machine hôte.

3.2.2 Les autres moyens de communication

L'application Java est actuellement organisée selon des `Subtypes` qui représentent chacun un mode de communication. Il en existe deux :

`Subtype_1` Sous-type de test permettant de vérifier que la communication entre TRNSys et l'application Java se fait correctement. Aucune donnée n'est échangée via ce sous-type.

`Subtype_2` Sous-type échangeant des objets Java sérialisés. En l'occurence, ce sont des `Map<String, Object>` où la clé correspond à une chaîne désignant l'objet associé.

Il faudra alors créer d'autres sous-types qui correspondront aux autres modes de communication.

Dans des perspectives de maintenance, d'abstraction et d'homogénéité, j'ai isolé plusieurs caractéristiques communes à certains types de communication. De plus, afin d'appliquer le principe de boîte noire relatif à la conception objet, je vais distinguer dans différents paquetages deux concepts identiques selon tous les types de communications : le formatage des données et la leur transmission.

Le formatage correspond au paquetage de classes chargées de formater les données avant de pouvoir les transmettre. De même, le principe est applicable dans l'autre sens pour obtenir un objet manipulable en fonction de ce qui a été reçu lors d'une transmition.

La transmission correspond au paquetageade classes chargées de la transmission des données selon le mode de communication choisi.

Le client comme le serveur manipuleront uniquement des objets de type `Map<String, Object>`, instancieront le formateur et le transmetteur sans se soucier du formatage ou de l'envoi de donnée mais uniquement du mode de communication. La figure 3.1 page 14 illustre ce principe pour le transfert des données du client vers le serveur.

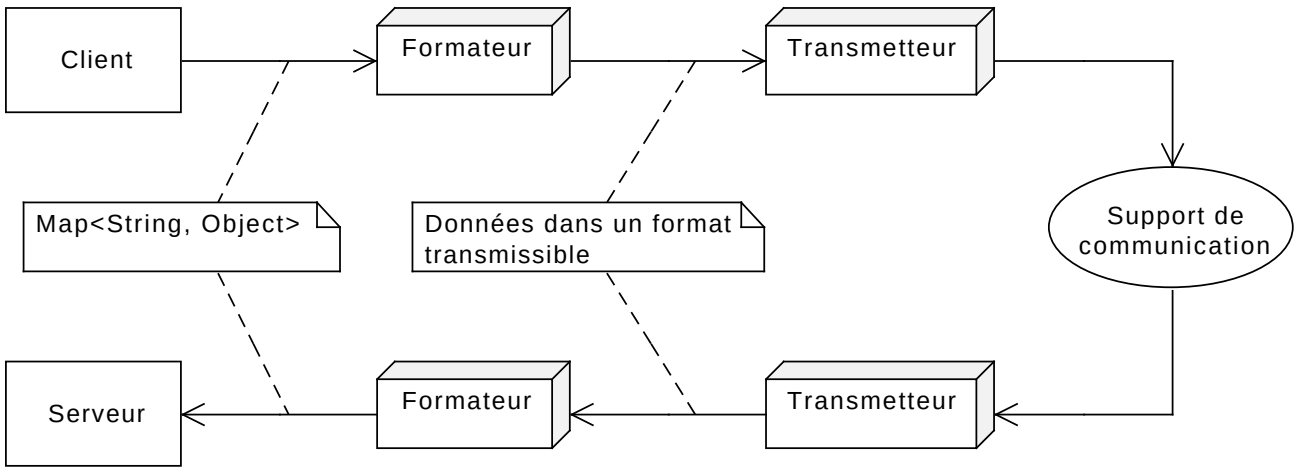

FIGURE 3.1 – Schéma de communication entre le client et le serveur

De plus, la partie d'administration réseau (initialisation, connexion, fermeture) du client sera la même pour tous les sous-types transmettant leurs données sur TCP/IP [6]. En effet, l'ouverture de socket et la connexion au serveur distant ne correspond pas à une communication Ivy [7]. C'est pourquoi, j'ai séparés les sous-types, non pas en les nommant

`Subtype_1` pour le sous-type de test.

`Subtype_2` pour l'échange d'objets sérialisés

`Subtype_3` pour l'échange de données au format XML

...

6. **T**ranmission **C**ontrol **P**rotocol over **I**nternet **P**rotocol. *Protocole de contrôle de transmission sur le protocole Internet*
7. Cette section est développée plus bas

`Subtype_n` pour le bus Ivy
mais plutôt
`Subtype_1` pour le sous-type de test.
`Subtype_2` pour les échanges TCP/IP
`Subtype_21` pour l'échange d'objets sérialisés
`Subtype_22` pour l'échange de données au format XML
...
`Subtype_3` pour le bus Ivy
Ainsi, cela nous ammenne à une structure de classe où le code produit est au plus haut niveau d'abstraction possible. De plus, qu'importe le nombre de sous-types d'échange sur TCP/IP que l'on ajoutera par la suite, les autres sous-types – Ivy dans notre cas – seront toujours nommés de la même manière (cf. figure 3.2 page 15).

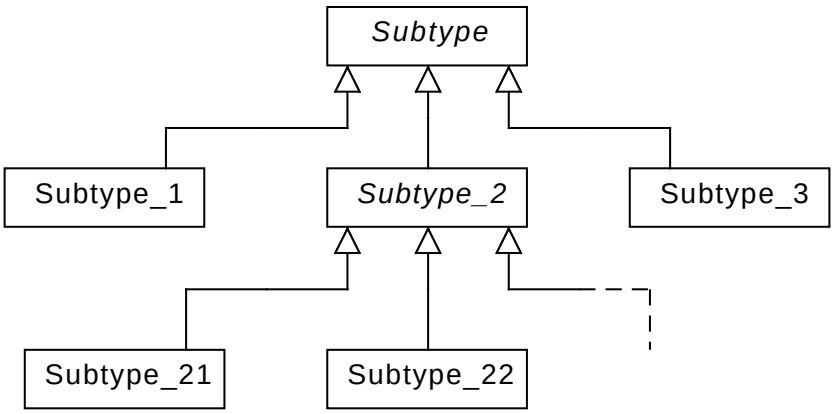

FIGURE 3.2 – Diagramme de classe des `Subtypes`

3.2.3 Recherches supplémentaires

Ivy

Ivy est un protocole simple et un ensemble de programmes et bibliothèques logicielles sous licence libre (LGPL [8]) qui permet aux applications de diffuser des informations via des messages de texte, avec un mécanisme de souscription basé sur des expressions régulières. Ivy est actuellement utilisé dans des projets de recherche dans le contrôle de traffic aérien et dans les communautés de recherche en IHM [9] aussi bien que dans des produits commerciaux.

Afin de ne pas encombrer la lecture de ce rapport, les détails concernant cette recherche sur le bus Ivy ont été placés à l'annexe C page 27.

oBIX

oBIX est un standard pour les interfaces de système de contrôle de bâtiment basé sur l'architecture de service web REST [10] – que l'ont peut assimilé au World Wide Web.

Dans la même optique que la section précédente sur le bus logiciel Ivy, les détails de l'étude concerannt oBIX ont été placés à l'annexe D page 29.

3.3 Mise en place de la solution

Pour mettre en place ce qui a été évoqué plus haut, il faudra d'abord mettre en place un environnement de travail fonctionnel mais aussi faire fonctionner le code existant. Cette étape consiste à recompiler la bibliothèque dynamique qui sera notre composant TRNSys – ceci sous environnement Windows (machine virtuelle). Je ne m'étends pas sur cet aspect de la réalisation car je n'y ai rien apporté. En effet, la bibliothèque était fonctionnelle à mon arrivée et quelques changements de configuration dans un lanceur personnalisé ont permis de faire fonctionner le sous-type de test. De plus, la partie sur la machine hôte ne nécessitait qu'un environnement de développement "classique". J'entends par là un environnement Eclipse, déjà mis en place depuis les projets réalisés lors de cette année universitaire.

8. **L**imited **G**eneral **P**ublic **L**icence. *Licence Publique Générale Limitée*
9. **I**nterface **H**omme **M**achine
10. **RE**presentational **S**tate **T**ransfer *Transfert d'Etat Représentatif*

3.3.1 Latence réseau

Trouver l'origine de cette latence revient à isoler la partie du code qui fait défaut et qui prend trop de temps. C'est pourquoi, j'ai décomposé le calcul de temps d'execution déjà présent en plusieurs calculs plus précis donnant des détails sur chaque appel plutôt que sur un pas entier de la simulation. Pour ce faire, j'ai donc utilisé le code suivant :

```
long begin = System.nanoTime();
// Fonction a chronometrer
long end = System.nanoTime();
```

Un simple calcul `end - begin = time` nous donne donc le temps d'exécution de la fonction concernée. Une première hypothèse serait de penser que ce qui pourrait prendre ce surplus de temps serait la sérialisation/désérialisation des objets Java. Le transfert réseau est mis hors de cause vu que nous sommes sur un réseau local, qui plus est entre machine hôte et machine virtuelle. Or après analyse des temps, il s'avère que c'est la réponse du serveur qui prend un temps trop important. En effet, la construction des `Map<String, Object>` prend un temps de l'ordre de la nanoseconde. Le transfert du client vers le serveur prend un temps de l'ordre de 10ms, ce qui est une latence normale pour un transfert sur un réseau local. Cependant, le transfert du serveur vers le client prend dans les 200ms. Malheureusement, après plusieurs tests, la

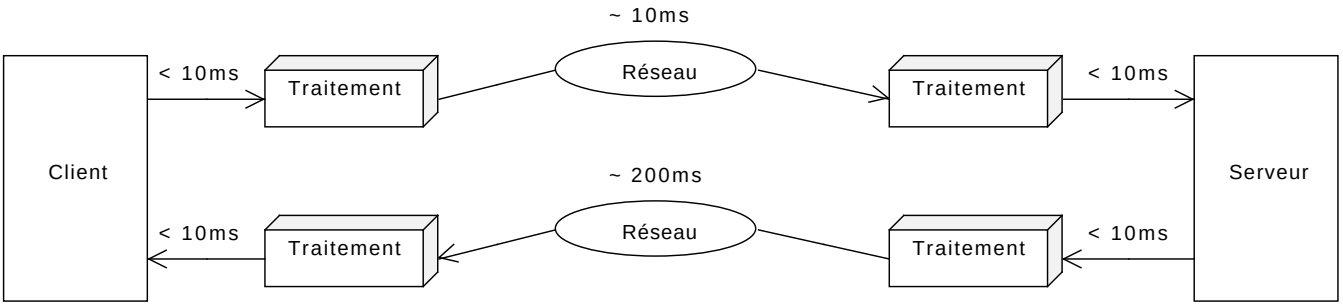

FIGURE 3.3 – Détail des temps calculés

latence réseau restait là sans aucun moyen de la réduire ni d'expliquer sa provenance. La taille des données transférées est sensiblement identique dans les deux cas (300 octets) et les temps de travail de part et d'autre sont normaux (< 10ms).

Sous les conseils de mon maître de stage, j'ai céssé de creuser la question, et j'ai commencé à implémenter les autres moyens de communciation de tranfert sur TCP/IP.

3.3.2 Les autres moyens de communication

Comme ennoncé dans le chapitre précédent, l'architecture choisie devrait facilité l'implémentation des différents types de communications via le réseau sur TCP/IP. Cette architecture est illustrée par le figure 3.4 page 17. L'architecture complète de l'application est disponible en annexe (E.1 page 32).

Les classes `Client` et `Server` ont toutes deux des attributs correspondant au formateur et au transmetteur génériques. La classe `Client` est utilisée par le sous-type `Subtype_2` pour initialiser la connexion réseau avec le serveur.
Il y a un foramteur et un transmetteur par mode de communication héritant d'une classe abstraite

Formateur

Les formateurs sont regroupés dans le paquetage `packetBuilders`. La classe `AbstractBuilder` est abstraite et fait office d'interface en fournissant les méthodes `handle` et `build`. Ces deux méthodes sont l'inverse l'une de l'autre.
`handle` prend en paramètre un `Object` reçu depuis le réseau et le transcrit en une `Map<String, Object>` manipulable par l'application.
`build` fait le même chemin à l'envers. Il transcrit une `Map<String, Object>` fournie par l'application et contenant les données à envoyer en un `Object` à transmettre sur le réseau.
Les `Objects` créés ou récupérés dépendent du mode de communication choisi.

Transmetteur

Les transmetteur suivent le même principe. Ils sont regroupés dans le paquetage `communicationHandlers`. La classe `AbstractTransmitter` est abstraite et sert d'interface fournissant les méthode `send` et `receive`. Les classes de ce paquetage s'occupent des échanges réseau en fonction du mode de communication choisi. Les objets sérialisés sont envoyés et reçus tels quels sans modification de la part des formateurs. Cependant, les données aux formats XML et

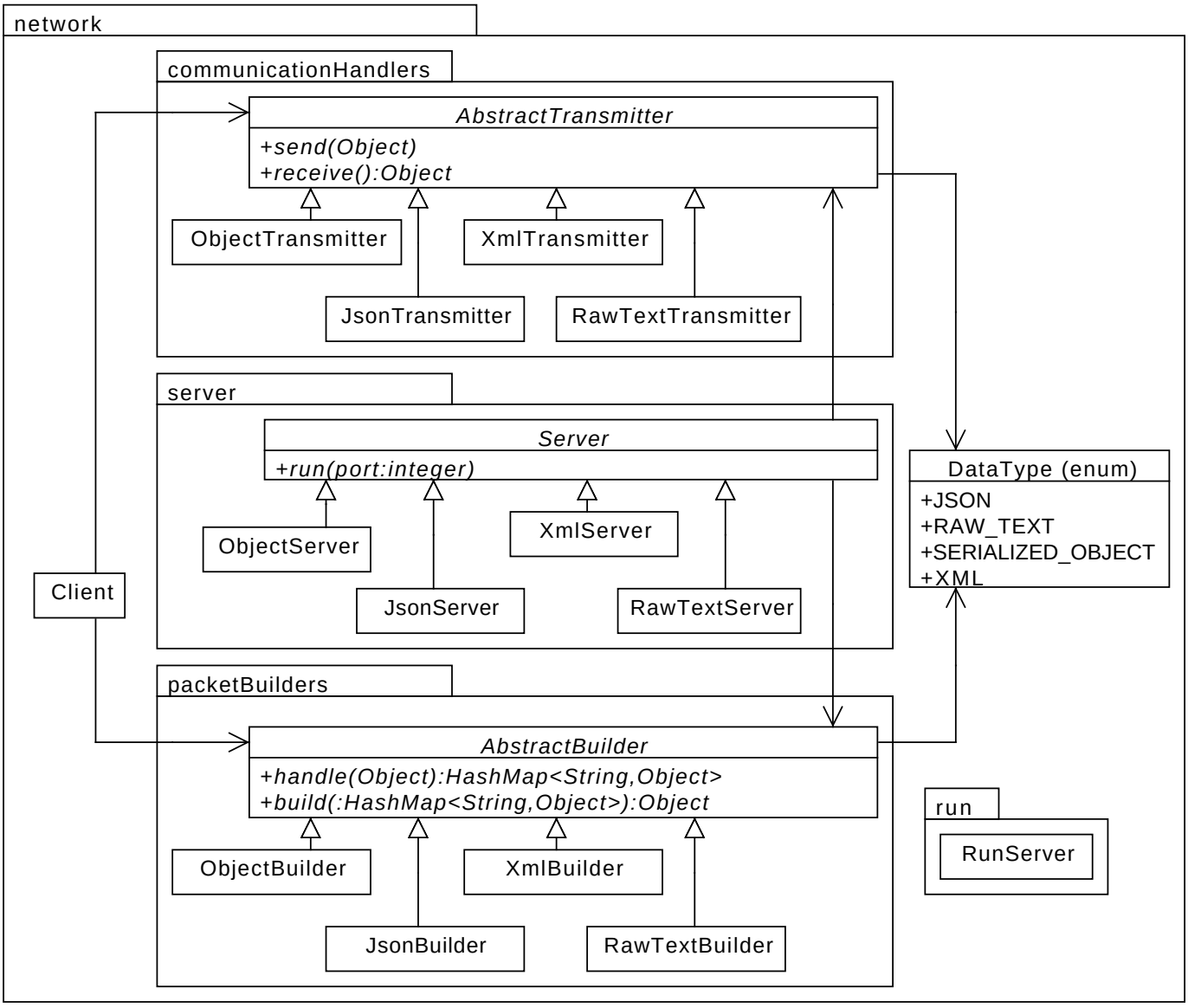

FIGURE 3.4 – Diagramme de classe du paquetage `network`

JSON sont envoyés au format texte brut, entraînant des reconstructions de `String` et un mode de lecture différent de celui utilisé pour la lecture d'objets sérialisés. C'est pourquoi, les deux méthodes `send` et `receive` travaillent avec des `Objects` pour garder la généricité.

Grâce à cette architecture et ce système de formateur et de transmetteur, pour implémenter un nouveau mode de communication via le réseau sur TCP/IP, il suffira de créer un nouveau formateur et un nouveau transmetteur, soit 4 méthodes uniquement.

Le premier type de communication que j'ai implémenté est le transfert de données au format XML. Après quelques tests de fonctionnement, j'ai remarqué que chaque pas était bien plus rapide pour les mêmes entrées/sorties qu'avec des objets sérialisés. Je me suis donc intéressé aux nouveaux temps de transfert. Gâce à l'échange en XML, je suis maintenant dans la fourchette des 10-30ms par pas de simulation. Le JSON étant moins verbeux que le XML, j'ai également calculé les temps de transfert avec ce mode de communication après un implémentation rapide due à l'architecture logicielle chosisie et à une bibliothèque Java – JSON-simple – qui permet de manipuler du JSON très facilement. La différence est moins significative mais tout de même perceptible. Les temps ne dépassent plus les 20ms. Après consultation avec François, nous avons décidé d'adopter ce mode de communication au vu des résultats qu'il fourni.

3.3.3 Ivy

Le `Subtype_3` correspond à la communication via le bus Ivy. Le concept de bus logiciel implique que le concept de client et de serveur ne s'applique pas. C'est pourquoi je parlerai d'émetteur et de récepteur ou d'écouteur. En effet, Ivy est un bus logiciel, c'est-à-dire qu'il donne l'illusion de communiquer via un bus physique. Lorsqu'une application se branche sur le bus, elle peut diffuser des données à toutes les applications qui sont elles aussi branchées sur le bus – ce sont les écouteurs. Par contre, chaque écouteur n'est pas forcément récepteur du message, il ne lui est pas forcément destiné. Physiquement, l'écouteur sait qu'il est récepteur du message selon un adressage numérique. Ivy fonctionne différemment sur ce point. les écouteurs souscrivent aux messages qu'ils désirent intercepter via des expressions régulières.

Cependant, il nous faudrait une syncronisation dans les échanges de messages. C'est pourquoi j'ai établi des messages correspondant à "Prêt à recevoir", "Prêt à envoyer" et "Envoi terminé". Ivy permet de souscrire à une expression régulière pour un seul mesage, cela va nous aider à syncroniser nos échanges. La figure 3.5 page 18 illustre une communication via Ivy durant un pas de la simulation. Les flèches qui bouclent avec 1 ou * représentent des expressions régulières auquels souscrivent les application respectivement une seule fois et plusieurs fois. Ainsi, ce système permet une communication

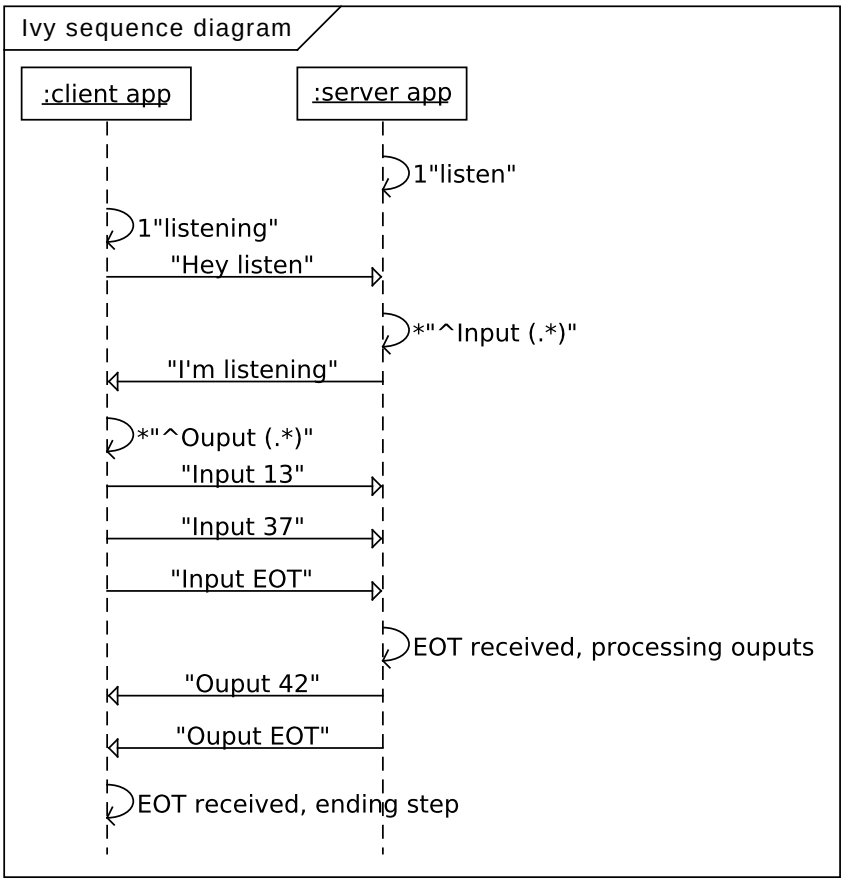

FIGURE 3.5 – Diagramme de séquence d'une communication via le bus Ivy durant un pas de simulation

synchrone entre les agents.

Chapitre 4

Bilan

4.1 Professionnel

Au terme de ce TER, les objectifs prédéfinis ont été atteints. Je suis même aller un plus loin en ce qui concerne Ivy et oBIX. Le travail que j'ai fourni offre à l'équipe un outil fonctionnel d'échange entre TRNSys et l'application Java selon plusieurs modes de transmission. De plus, l'architecture choisie permet l'ajout de nouveaux modes de communciation de façon rapide et pauvre en écriture de code. La partie de l'application utilisant Ivy est opérationelle mais peut ête encore adaptée pour correspondre pleinement aux besoin qui sont l'échange de données principalement numériques. En revanche, rien ne permet actuellement à l'application de communiquer avec des périphériques suivant le standard oBIX. Cependant, j'espère que mes recherches et précisions sur le sujet permettront à l'équipe de continuer sur cette voie. L'application est maintenant fonctionelle et a un avenir certains au sein de ce projet et peut-être au sein d'autes projets de l'équipe.

4.2 Personnel

Cette expérience m'a apporté d'un point de vue technique mais aussi m'a renseigné sur mes capacités à travailler en autonomie. En effet, que ce soit à l'IUT, pour mon stage de fin de DUT ou tout au long de mon année de L3, j'ai fais la plupart de mes TP ou projets en collaboration plus ou moins étroite avec d'autres personnes, étudiants ou professionnels de l'informatique. C'est pourquoi cette mission m'a permis de savoir si j'étais capable de travailler seul. Malgré quelques petites faiblesses je dois bien l'avoué, je pense que je sais trouver la motivation et l'envie d'avancer dans un projet pour peu que celui-ci m'intéresse et m'apporte quelque chose. Ce projet m'a fait découvrir le monde de la recherche, l'organisation des équipes, des exemples de sujet de recherche. Les discutions que j'ai eu avec d'autres doctorants autres que mon maître de stage ont été très enrichissantes sur l'ambiance dans ce milieu qui ne nous est pas familier.

Pour la partie compétence informatique, j'ai essentiellement réutiliser des compétences que je possédais déjà. J'ai par contre découvert de nouveaux concepts qui m'ont ouvert l'esprit à d'autres méthodes de développement. Les échanges via bus logiciel m'offrent de nouvelles perspectives de développement d'applications distribuées. Le standard oBIX pour la domotique tend à me faire penser que celà existe dans d'autres domaines que je côtoie peut-être plus souvent sans même le soupçonner. Un bilan très positif sur le plan personnel qui me conforte dans mon choix de poursuivre vers du développement logiciel de plus en plus approfondi et par conséquent, de plus en plus intéressant.

Annexe A

Glossaire

Bibliothèque dynamique C'est un fichier binaire contenant du code pouvant être appelé par un proramme tiers. Les bibliothèques dynamiques sont appelées durant l'exécution d'un programme de façon dynamique, c'est-à-dire quand le programme en a besoin.

Routeur Un routeur est un périphérique réseau via lequel il est possible d'interconnecter des machine sur un réseau local. Le routeur sert également de passerelle pour se connecter à un autre réseau englobant comme Internet.

FAI Un Fournisseur d'Accès Internet est une firme qui permet à un utilisateur d'accéder à l'ensemble du réseau Internet.

EDI Un Environnement de Développement Intégré est un ensemble d'outils regroupés dans un même logiciel facilitant le développement d'applications informatiques.

Dépôt Debian Un Dépôt officiel Debian est un serveur sur lequel il est possible de récupérer des archives permettant l'installation de programme sur la distribution Debian. Les dépôts officiels garantissent la vérification des logiciels téléchargeables.

Git C'est un logiciel de versionnement. C'est-à-dire un logiciel permettant de conserver plusieurs versions d'un espace de travail collaboratif afin de revenir à une version qui fonctionne si quelque chose tourne mal.

GitHub C'est une platerforme de service web permettant de créer des dépôts Git distants et qui fournit tout un tas de statistiques sur l'activité de ces dépôts (nombre de modif par jour/mois/année, etc.).

Wiki Un wiki est un site web dont les pages sont modifiables par les visiteurs afin de permettre l'écriture et l'illustration collaborative des documents numériques qu'il contient.

UML C'est une norme de modélisation logicielle afin d'améliorer la conception et éviter des problèmes lors de l'écriture de code.

JNI C'est un bibliothèque java qui permet d'appelé facilement du code Java depuis du code natif et vice-versa.

Sérialisation d'objet C'est le procédé qui permet à un langage orienté-objet d'enregistrer des objets de manière permanente ou de les envoyer via le réseau.

XML C'est un langage textuel de représentation de données sous forme de balises.

JSON C'est un langage textuel de représentation de données, spécialement adapté à la représentation d'objet dans les langages oreientés-objet.

Socket Une socket correspond à un point de connexion avec le réseau représenté par une adresse IP et un port.

TCP/IP C'est un norme de communucation de la couche transfert du modèle OSI. Elle établi des connexions en mode connecté et permet d'éviter la perte de trames.

LGPL C'est une licence dite libre dérivée de la Licence Publique Générale GNU. Elle est utilisée pour des logiciels où une publication entièrement libre de toute l'offre est impossible.

Expression régulière C'est une chaîne de caractère qui suit certaines conventions et sert de filtre applicables à d'autres chaînes de caractères pour établir des correspondances.

Annexe B

Références

Documentation Git http://git-scm.com/documentation

Documentation Ivy http://www.eei.cena.fr/products/ivy/

Documentation Ivy (Javadoc) http://www.eei.cena.fr/products/ivy/documentation/ivy-java-api/fr/dgac/ivy/package-summary.html

Documentation oBIX http://www.obix.org/

GitHub https://github.com/

Framabook LaTeX http://framabook.org/latex.html

Wikipedia http://fr.wikipedia.org

Annexe C

Ivy, le bus logiciel

Ivy est un bus logiciel qui a été conçu au CENA [1] et qui est développé depuis 1996.

Un bus logiciel

Un bus logiciel est un système qui permet aux applications d'échanger des informations avec l'illusion de les diffuser globalement. L'acceptation ou l'ignorance de ces messages est basée dur le contenu de ces messages. D'autres dénominations existent comme "service de publication-souscription à des notifications" ou "Logiciel d'échange d'information basé sur des messages". Les bus logiciels regroupent plusieurs logiciels écrit dans différents langages de programmation sur différentes plateformes.

À l'intérieur d'une même application, utiliser un bus logiciel tel qu'Ivy est comme utilisé le système de gestion des événements d'une bibliothèque graphique. D'un côté des messages sont émis sans se soucier de qui ils concernent. De l'autre côté, chaque application décide de traiter un message s'il a un certains type ou s'il suit un certains format. Les bus logiciels sont principalement appréciés pour le développement de nouvelles applications – appelées agents – et à administrer dynamiquement une collection d'agents hétérogènes. De nouveaux agents se connectent au bus, envoient ou recoivent des messages, et quittent le bus sans bloquer les autes agents.

Plusieurs bibliothèques dans plusieurs langages implémentent Ivy. Elles répondent toutes au caractéristiques de base de gestion du bus.

- Se connecter au bus
- Envoyer un message
- Lier un format de message sous forme d'expression régulière à une fonction précise

La biblitohèque Java fournit aussi quelques fonction qui facilite la programmation.

- Lier un format de message à une fonction pour un seul message et délier ensuite.
- Attendre un message particulier
- Attendre un client particulier

Une application en console est fournie avec Ivy. Elle s'appelle IvyProbe et permet de se connecter à bus et de souscrire à un format de message. La principale utilité de cet outil est de surveiller ce qu'il transite sur le bus en souscrivant à l'expression régulière `"(.*)"` qui correspond à n'importe quelle chaîne de caractère.

1. **C**ente d'**É**tudes de la **N**avigation **A**érienne

Annexe D

oBIX, open Building Information eXchange

oBIX est un standard pour des services web basé sur l'architecture REST servant d'interface à des systèmes de régulation dans un bâtiment. oBIX permet de lire et écrire des données à travers un réseau de composants en utilisant le format XML et des URIs[1] à l'intérieur d'un système conçu spécialement pour de la domotique. Du fait qu'il suit l'architecture REST, il peut être comparé au World Wide Web qui suit aussi cette architecture. Par analogie, si le WWW est une toile où chaque nœud est une page indépendante ou reliée à d'autres nœuds via des hyperliens, un réseau oBIX est une toile où les nœuds sont des objets indépendants ou reliés à d'autres nœeuds via des références à ces autres nœuds en utilisant leurs URIs.

Fonctionnement

Chaque objet est désigné par une URI. Toujours à l'instar du WWW, oBIX suit le principe de client/serveur. Les requêtes oBIX sont assez semblables aux requêtes HTTP[2]. En effet, seuls trois types de requêtes existent dans le standard oBIX :

Lecture Récupère l'état d'un objet sur le serveur distant

Écriture Modifie l'état d'un objet sur le serveur distant

Invocation Appelle une méthode distante avec ou sans paramètres d'entrée et retourne l'objet en sortie (s'il y en a un).

Il est d'ailleurs d'usage d'utiliser des requêtes HTTP pour effectuer des requêtes oBIX. La tableau D.1 page 29 fait le lien entre les requêtes HTTP et leur interprétation oBIX.
En complément, la figure D.1 page 29 montre les diagramme de séquence d'appel de ces trois requêtes.

Requête oBIX	Requête HTTP	Cible de la requête
Lecture	GET	Un objet via son URI
Écriture	PUT	Un objet modifiable via son URI
Invocation	POST	Un objet correpondant à une méthode via son URI

TABLE D.1 – Tableau de correspondance ente les requêtes HTTP et leur interprétation oBIX

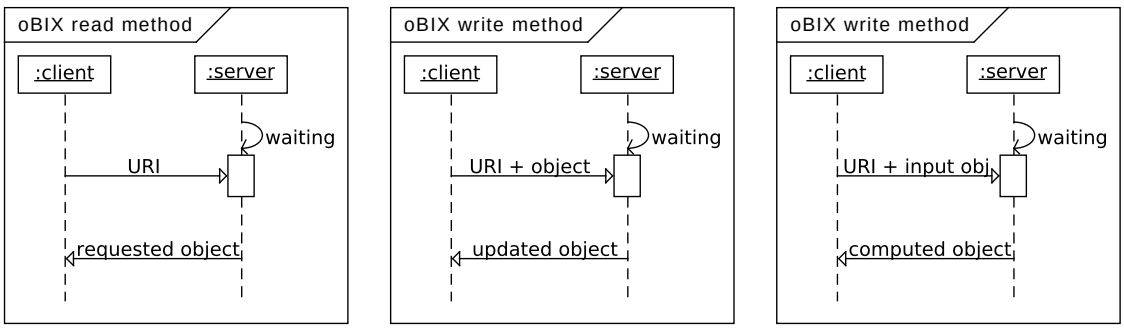

FIGURE D.1 – Diagrammes de séquence des requêtes oBIX

1. **U**niform **R**essource **I**dentifier. *Indentificateur de ressource uniforme*
2. **H**yper**T**ext **T**ransfert **P**rotocol. *Protocole de tranfert d'hypertexte*

Annexe E

Architecture logicielle

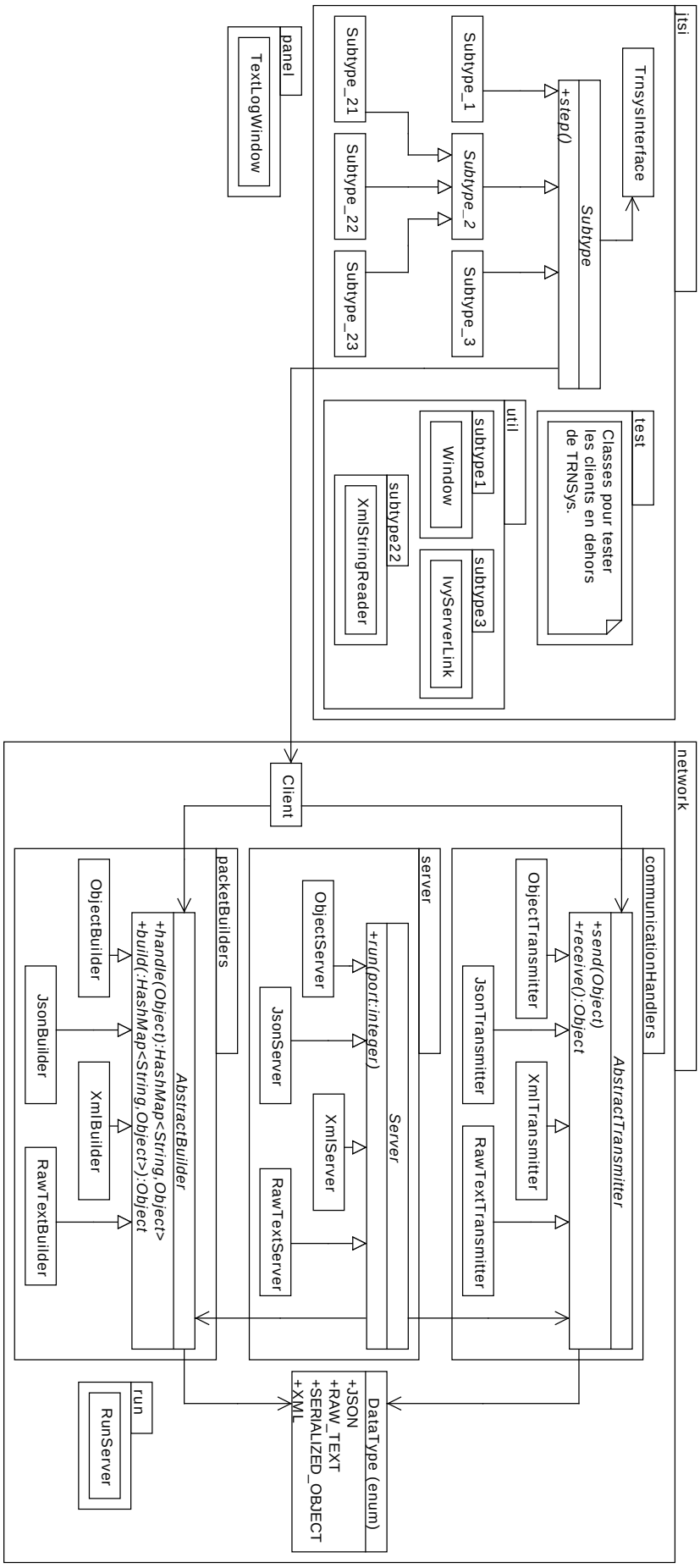

FIGURE E.1 – Architecture complète de l'application. La plupart des attributs et méthode ne gênant pas la compréhension ont été masqués pour des raison de lisibilité