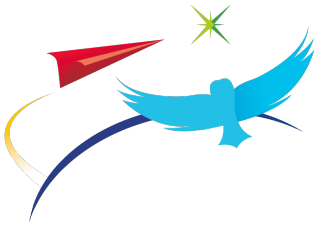


Institut Supérieur de l'Aéronautique et de l'Espace

PIR 2ème année Supaero 2009

Simulation distribuée avec HLA Appliquée à FlightGear

Vincent Lecrubier
Benjamin Sotty

Encadrants :
Pierre Siron
Régine Leconte

Table des matières

Introduction

Pendant ce PIR nous avons étudié le fonctionnement de simulation distribuées soumis à la norme HLA-High Level Architecture, implémenté par CERTI dans le but de pouvoir l'appliquer une telle simulation pour le vol en patrouille de plusieurs simulateurs de vols Flight Gear. Nous verrons dans une première partie l'intéret de l'utilisation d'une simulation distribuée soumis à la norme HLA, et nous détailerons cette norme. Ensuite nous verrons comment implémenté une fédération sous CERTI, nous décrirons pour cela la structure et le cycle de vie d'une fédération TestVL que nous avons implémenté. Enfin nous décrirons le plugin Flight Gear-HLA et nous verrons comment le modifier en se servant de notre travail pour aboutir au vol en patrouille de plusieurs Flight Gear.

1 Simulation distribuée et HLA

1.1 La simulation

La simulation est une application de l'informatique permettant d'observer le comportement d'un système que l'on a modélisé, en fonction de scénarii que l'on définit. Pour modéliser un système, nous le décrivons par un ensemble de variables d'état, d'événements et d'entités :

- Une variable d'état est une variable caractérisant un attribut du système à simuler, comme l'altitude d'un avion, le niveau du stock de matières premières d'une usine, la position d'une bille… Nous appelons l'état du système l'ensemble des variables d'état du système.
- Un événement est une occurrence à un point donné dans le temps susceptible de modifier l'état du système. Par exemple, la panne d'un moteur d'avion, l'utilisation de matières premières par la chaîne de montage d'une usine et la collision de deux billes sont des événements.
- Une entité est un objet qui a une existence propre dans le système, comme le pilote d'un avion, un ordre de production dans une usine, un client, … Souvent un événement est associé à une ou plusieurs entités. Par exemple l'événement « collision » correspond à une interaction entre deux entités « bille ».

Un exemple de simulation, que nous retrouverons le long de notre travail serait la simulation d'un billard.

- Les variables d'état seraient les position des différentes billes ainsi que leur vitesse.
- Les évenements correspondraient aux interactions entre billes.
- Les entités seraient des billes.

1.2 Intérêt de la simulation

La simulation a deux grands domaines d'application :

- La prédiction, l'aide à la décision lors de choix de conception, elle permet d'évaluer différentes décision afin de choisir la meilleure, dans un but d'optimisation du produit. Elle peut entre autre remplacer le prototypage, avec à la clé des économies parfois très importantes pour un résultat identique. Dans ce cadre, la simulation permet aussi d'appréhender le problème plus en profondeur et peut faire émerger de nouveaux concepts.
- L'entrainement, qui a un coût important s'il est effectué en réel, alors que la simulation permet souvent un entrainement presque aussi efficace. Une heure d'entrainement sur simulateur coûte généralement de 2 à 10 fois moins cher qu'un entrainement réel, si celui ci est possible. Mais on considère qu'une heure d'entrainement sur simulateur équivaut en terme d'expérience à entre 0.5 et 1heure d'entrainement réel. Il est donc facile de comprendre l'intérêt de la simulation pour l'entrainement, que ce soit celui des pilotes, des contrôleurs aériens ou dans d'autres milieux comme la formation des opérateurs en centrale nucléaire.

La simulation prend aujourd'hui une part de plus en plus importante dans les processus de conception industrielle. Plusieurs facteurs sont à l'origine de cet essor :

- La mondialisation et ses effets sur la concurrence : aujourd'hui, sur des marchés de plus en plus concurrentiels, la marge de manœuvre des industriels est de plus en plus réduite, les produits doivent être optimisés au maximum afin d'atteindre la rentabilité. Or l'optimisation n'est réalisable correctement que si elle est effectuée au début du cycle de développement, et seule la simulation

permet à cette étape d'évaluer les différents choix possibles.

- Le coût des essais réels est non négligeable, et l'exigence de rentabilité ne permet plus de s'offrir de nombreux prototypages dans le monde réel.

- Le progrès des nouvelles technologies permet d'effectuer des simulations de plus en plus précises et nombreuses à coût égal, rendant cette méthode attrayante.

- Dans de nombreux domaines, la formation et l'entrainement du personnel a un coût qui s'envolerait si la simulation n'existait pas, on pense par exemple à la formation des pilotes, des contrôleurs, etc...

1.2.1 Simulation distribuée

Les simulations de systèmes complexes étant parfois très lourdes, il est souvent devenu inenvisageable de construire une simulation complète tournant sur une seule machine. Les coûts en matériel seraient trop importants, alors que l'on bénéficie d'un effet d'échelle lorsque l'on répartit la simulation sur de nombreuses machines. De plus, les progrès dans le domaine des machines multiprocesseurs et des réseaux de communication permettent aujourd'hui de réaliser des simulations distribuées très efficaces.

Enfin, on remarquera que de nombreux systèmes complexes peuvent être divisés en plusieurs systèmes plus simples, faiblement couplés. Ainsi, on profite de cette division pour répartir les différents éléments de la simulation, chaque sous système étant simulé individuellement. Les différentes parties de la simulation s'échangeant des messages correspondant à leurs interactions. Moins ces interactions sont nombreuses, plus la distribution de la simulation est efficace, car les différents élément n'ont alors que peu de messages à s'échanger.

1.2.2 Différents modèles de simulation distribuée

De nombreux modèles de simulation distribuée ont été réalisé depuis que ce concept existe. La plupart de ces modèles ne sont pas généralistes et sont optimisés pour un domaine d'application particulier.

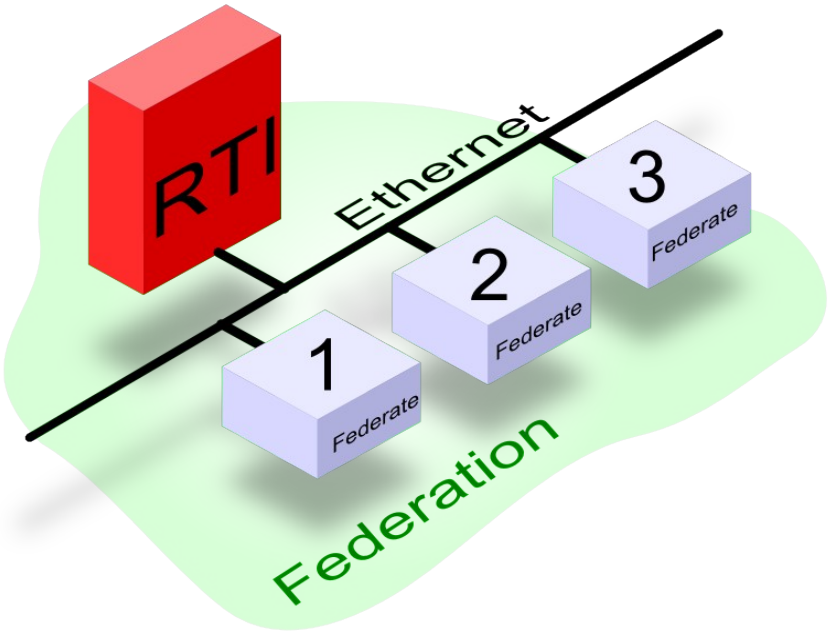
Nous allons décrire succinctement les principales architectures de simulation distribuée qui ont existé avant HLA. Nous verrons SIMNET (*SIMulation NETworking*), ALSP (*Aggregate Level Simulation Protocol*) et DIS (*Distributed Interactive Simulation*).

A partir de ces différentes architectures, le DoD a identifié le besoin de mettre en place une normalisation sur les méthodes et sur les outils de modélisation et d’architecture de simulations distribuées. Ce qui a mené a la mise en place du standard HLA (High Level Architecture).

1.3 Le standard HLA

1.3.1 Notions générales

HLA est un standard de simulation distribuée, définissant les différents éléments d'une simulation distribuée, ainsi que leurs interactions. Une simulation en HLA est appelée fédération. Elle est constituée de différents éléments. Un schéma très général est proposé ci dessous :



On voit donc qu'une fédération HLA est constituée :

•De plusieurs fédérés, qui sont des simulations représente une partie de la simulation globale, un sous système du système complet simulé. Les fédérés sont fortement liés à la simulation, ils représentent un élément particulier du système global. Cependant, un fédéré peut être réutilisé dans plusieurs simulations, par exemple si on souhaite simuler deux systèmes différents, mais comprenant un sous système identique. C'est le premier avantage de HLA.

•D'une unique RTI (RunTime Infrastructure), qui organise la fédération et route les messages que s'envoient les fédérés. Le deuxième avantage de HLA est que cette RTI, qui effectue une part importante du travail, est très générale, et est réutilisable pour tout types de simulations. La lourde charge de travail liée à la programmation du routage des messages dans une simulation distribuée complexe est donc quasiment supprimée.

Le standard HLA est constitué de 4 éléments :

•Les **règles** qui définissent les responsabilités des fédérés et de la fédération.

•L'**OMT** (Object Model Template), qui fournit une standardisation de la documentation du modèle objet HLA. Il s'agit d'un formalisme de documentation orienté objet pour les fédérés et les fédérations sous forme de « Trames de Modèles Objets » (OMT).

•Les spécifications de l'interface de programmation (**API**). Cette API (ApplicationProgramming Interface) est un ensemble de fonctions bas niveau, bien documentées, permettant de programmer des applications de haut niveau. L'API HLA décrit les services de la RTI que les fédérés utilisent pour communiquer entre eux au sein d'une fédération.

•Le **FEDEP** (Federation Development and Execution Process) qui offre un certain nombre de recommandations pour la conception de fédérations HLA.

1.3.2 Les règles de HLA

Cinq règles concernent la fédération et cinq concernent les fédérés.

1.Obligation du respect du formalisme OMT de HLA pour décrire les fédérations

Les fédérations auront un modèle objet (FOM : Federation Object Model), documenté

conformément à l'OMT.

2.Représentation des objets au niveau des fédérés et non de la RTI

Au sein d'une fédération, toutes les représentations d'objets se situeront au niveau des fédérés, et non au niveau de la RTI.

3.Passage par la RTI pour les échanges de données publiques entre les fédérés

Pendant l'exécution d'une fédération, tous les échanges de données figurant dans la FOM et agissant entre fédérés devront s'effectuer via la RTI.

4.Respect de la spécification d'interface pour l'utilisation de la RTI

Pendant l'exécution d'une fédération, les fédérés devront interagir avec l'infrastructure d'exécution (RTI) en accord avec les spécifications d'interfaces de HLA.

5.**A tout moment pendant l'exécution d'une fédération, un attribut n'appartien qu'à un fédéré**

Pendant l'exécution d'une fédération, un attribut d'une instance d'objet ne sera la propriété que d'un seul fédéré.

6.Obligation du respect du formalisme OMT de HLA pour décrire les fédérés

Les fédérés auront un modèle objet de simulation (SOM : Simulation Object Model), documenté conformément à l'OMT.

7.**Transparence des attributs contrôlés par un fédéré et capacité d'interactions d'après leur SOM**

Les fédérés seront capables d'envoyer et/ou de recevoir des valeurs d'attributs d'instances d'objets ainsi que d'envoyer et/ou recevoir des interactions conformément au SOM.

8.**Capacité des fédérés à transférer et/ou à accepter le contrôle d'attributs d'après leur SOM**

Les fédérés seront capables de transférer ou d'accepter le contrôle d'attributs dynamiquement pendant l'exécution de la fédération, conformément au modèle objet de simulation (SOM).

9.**Faculté des fédérés à faire varier les conditions de mise à jour des attributs d'après leur SOM**

Les fédérés seront capables de faire varier les conditions (exemple : seuil) sous lesquelles ils fournissent des mises à jour pour les attributs publics des objets, conformément à leur modèle objet de simulation (SOM).

10.Aptitude des fédérés à gérer un temps local, selon les principes décrits dans HLA

Les fédérés seront capables de gérer un temps local de telle sorte que cela leur permette de coordonner les données échangées avec les autres membres d'une fédération. Les spécifications seules ne permettent pas de générer un système de simulation distribuée, la RTI représente son implémentation, c'est le moteur de HLA.

1.3.3 L'Object Model Template

Une fédération est décrite par un FOM (Federate Object Model). Le FOM contient la description de différents items :

- Des objets, qui correspondent à la notion d'objet en programmation, et correspondent aux objets modélisés dans la simulation. Les objets peuvent hériter, mais l'héritage multiple n'existe pas. Exemple d'objet : Bille.
- Les attributs de ces objets. Exemple d'attribut : Position d'une bille.

•Des interactions, qui correspondent aussi d'une certaine manière à la notion d'objet en programmation, mais qui correspondent aux différentes interactions possibles entre les objets de la simulation. Exemple d'interaction : Collision.

•Les paramètres de ces interactions. Exemple de paramètre d'interaction : Intensité d'une collision.

•Des espaces, qui correspondent à différents espaces, au sens mathématique, utiles lors de la simulation. Exemple d'espace : Position géométrique.

•Des dimensions, qui définissent ces espaces : exemple de dimension : Donnée selon l'axe X d'une position géométrique.

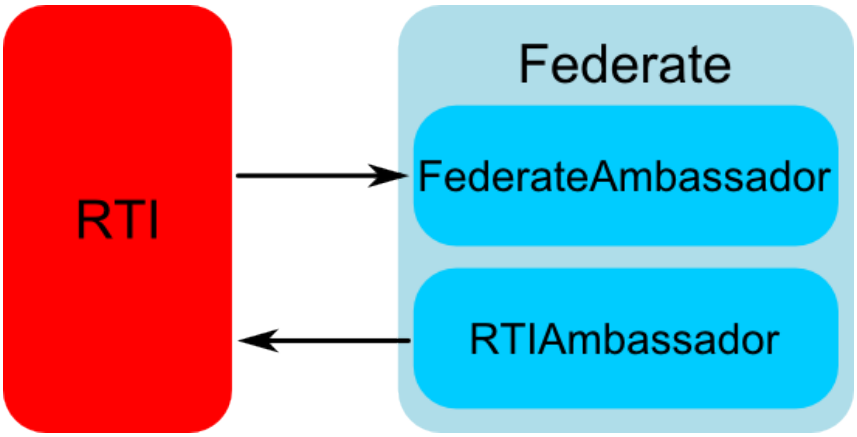
Le FOM est bien entendu très lié à la simulation, ce FOM est utile à tous les éléments de la simulation, il est représenté concrètement par le fichier .fed, propre à chaque fédération.

Un exemple de fichier .fed est présenté ci dessous et permet de retrouver les différents attributs présentés plus haut.

```
;; Exemple

(Fed
(Federation Test)
  (FedVersion v1.3)
  (Federate "fed" "Public")
  (Spaces
    (Space "Geo"
      (Dimension X)
    )
  )
  (Objects
    (Class ObjectRoot
      (Attribute privilegeToDelete reliable
                          timestamp)
      (Class RTIprivate)
      (Class Bille
        (Attribute PositionX RELIABLE TIMESTAMP)
      )
    )
  )
  (Interactions
    (Class InteractionRoot BEST_EFFORT RECEIVE
      (Class RTIprivate BEST_EFFORT RECEIVE)
      (Class Collision RELIABLE TIMESTAMP
        (Sec_Level "Public")
        (Parameter Intensite)
      )
    )
  )
)
```

1.3.4 L'API HLA



Les deux classes importantes de L'API sont :

•RTIAmbassador : qui fournit au fédéré des méthodes permettant de transmettre des messages et de mises à jour à l'ensemble de la fédération via la RTI

•NullFederateAmbassador. : qui est une classe virtuelle comprenant des méthodes appellés « Callbacks », qui doivent être implémentées par le dévellopeur. Ces méthodes définissent les actions du fédéré lorsqu'il reçoit des messages de la part d'autres fédérés via la RTI. La RTI provoque l'appel de ces méthodes, et le développeur doit créer le code afin de générer un comportement adapté.

Généralement un fédéré est représenté par une classe héritant de NullFederateAmbassador, et implémentant les méthodes virtuelles appellées « Callbacks ». Cette classe dispose d'un attribut de type RTIAmbassador, et elle appelle les méthodes de RTIAmbassador pour envoyer des messages.

Un prototype de classe représentant un fédéré serait donc :

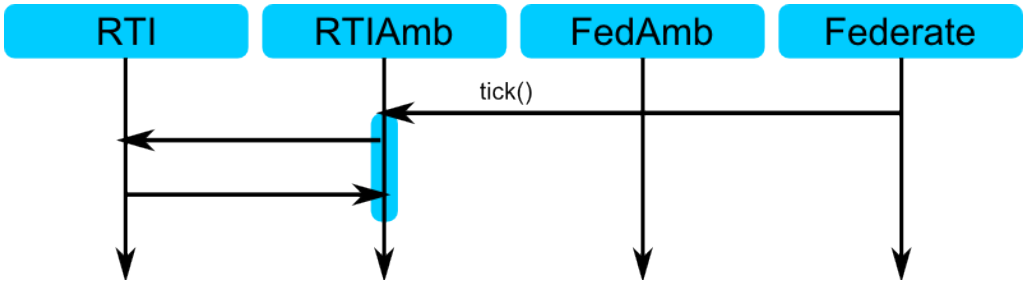
```
class Federate : public NullFederateAmbassador
{
  public:
    Federate();
    virtual ~Federate();
    // Callbacks
    void announceSynchronizationPoint(...);
    void federationSynchronized(...);
    void timeAdvanceGrant(...);
    void discoverObjectInstance();
    void reflectAttributeValues(...);
    void receiveInteraction(...);
    void removeObjectInstance(...);
    void sendInteraction(...);
    void sendUpdate(...);

  protected:
    RTI::RTIambassador rtiamb ;
};
```

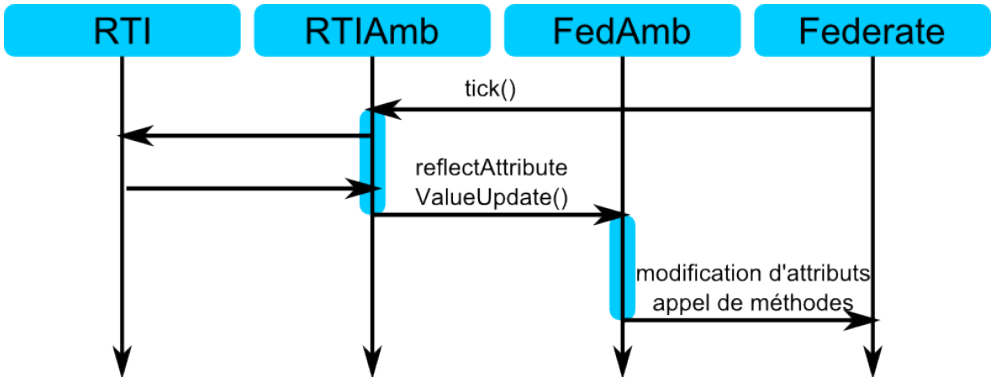
Pour illustrer le rôle de ces deux classes, nous allons étudier deux situations typiques : La réception

de mises à jour d'attributs par un fédéré, et l'envoi de mise à jour d'attributs par un fédéré.

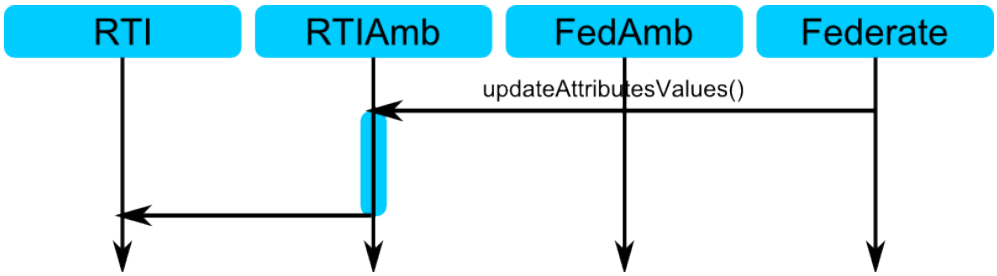
Dans le premier cas, le fédéré suit la démarche suivante si aucune mise à jour n'est disponible. Il envoie un tick et rien ne se produit.



Si une mise à jour est disponible, le fédéré envoie un tick, qui provoque une mise à jour de ses données.



Pour envoyer ses attributs, un fédéré utilise une méthode de son RTIAmbassador, qui envoie ses données à la RTI.



1.3.5 Le FEDEP

Le FEDEP (Federation Development and Execution Process) est un outil méthodologique de conception générique d'une fédération HLA. Le FEDEP est défini dans la norme HLA et chaque développeur souhaitant réaliser une simulation avec HLA devrait suivre cette méthodologie afin d'éviter les écueils. Cette méthode se résume en 6 étapes :

1.Définir les objectifs de la fédération

L'équipe de développement et les utilisateurs conçoivent ensemble des documents définissant es objectifs et besoins de la fédération à réaliser.

2.Développer un modèle conceptuel de fédération

Cela consiste à générer des scénarii en utilisant une base de données scénario ou en développant ces scénarii à partir d'outils. Le développement conceptuel passe par la définition de l'environnement de la fédération (les besoins des utilisateurs, les objectifs de la fédération …), la traduction de ces

objectifs en activités sous forme d'entités et actions à intégrer dans la fédération, la validation des principes de la fédération et la proposition d'un schéma d'implémentation. Outre le développement du scénario de la fédération, on effectue donc une analyse conceptuelle ainsi qu'une définition plus détaillée des besoins.

3.Structurer la fédération

Cette étape consiste à choisir les fédérés et leurs fonctions d'après les scénarii élaborés afin de préparer un projet de fédération sous forme de guide au développement, au test et à l'exécution d'une fédération. OML (Object Model Library [98S-SIW-142]) fournit un ensemble d'objets utiles pour la construction de la fédération. Il permet aussi de sauvegarder et restituer les objets définis dans les SOMs et FOMs.

4.Développer la fédération

Le but, dans cette phase, est de mettre en place le FOM, de vérifier la validité de la fédération en termes d'interopérabilité, de faisabilité …, d'implémenter les modifications des fédérés pour la phase suivante d'intégration et de test de la fédération. Outre la mise en place du FOM, on établit les règles d'exécution de la fédération et on développe ou modifie les fédérés participant à la fédération.

5.Intégrer et valider la fédération

L'objectif est de vérifier que tous les fédérés peuvent interopérer pour accomplir les objectifs de la fédération. Il y a la production du FEPW (Federation Execution Planners Workbook) qui générera les performances exigées de la fédération, ceux des fédérés et de la qualité du réseau : document indispensable avec le scénario et le FOM pour réaliser le test de la fédération.

Les tests se déclinent en trois niveaux :

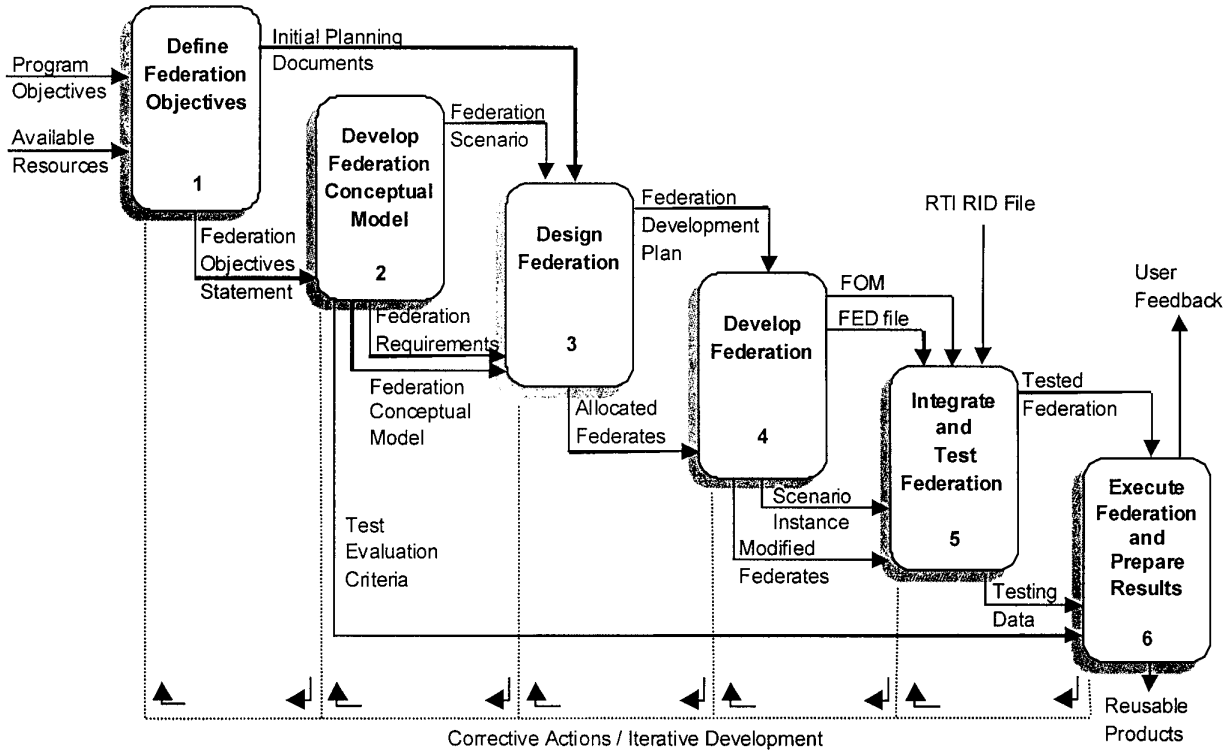
◦Test Fédéré, vérifiant que chaque fédéré est correctement implémenté d'après le FOM et le FEPW.

◦Test Intégration, analysant si les fédérés communiquent bien avec la RTI et échangent les données comme prévu dans le FOM.

◦Test Fédération, jugeant s'il y a eu une correcte interopérabilité entre fédérés d'après le parcours du scénario, validant les objectifs de la fédération.

1.Exécuter la fédération et préparer les résultats

Ce dernier cycle permet d'exécuter la fédération, de rendre des résultats dérivés d'après les données brutes issues de l'exécution et de préparer les résultats qui seront analysés pour déterminer s'ils respectent les objectifs de la fédération (conformité avec FOM, SOM …) : dans ce cas, on les stockera ; sinon on reviendra aux phases initiales, en corrigeant les points en erreur.



6 Etapes Traitements FEDEP

1.3.6 CERTI

HLA est une spécification, et non pas un logiciel, en conséquence, il existe plusieurs implémentations de HLA. L'implémentation qui nous intéresse s'appelle CERTI et elle a été développée au CERT depuis 2004. CERTI est une implémentation libre et performante de la spécification HLA.

2 Implémentation d'une fédération HLA avec CERTI

Pour comprendre comment est implémenté un code qui utilise HLA, outre les documentations papiers, nous nous somme appuyé sur les sources de deux programmes de test fournis avec CERTI : Create_and_Destroy et Billard. Puis nous avons créer notre propre programme en implémentant la classe TestVL, dont la simulation ne consiste qu'en la modification d'une chaîne de caractères. Cette application simple avec ces commentaires et le document présent pourront servir de base pour comprendre l'implémentation d'un code de fédéré, en vu de la modification du plugin de Flight Gear.

2.1 Structutre de l'application Essai1

Notre programme se compose des fichiers TestVL.cc et TestVL.hh qui implémentent la classe TestVL, d'un fichier Essai1.cc qui contient le main de la simulation et d'un fichier TestVL.fed. Vous trouverez en annexe 1 la documentation des méthodes de CERTI utilisée, en annexe 2 les codes sources commentés des fichiers TestVL.cc et TestVL.hh et le fichier TestVL.fed. La figure ci-après présente le diagramme UML de la classe TestVL.

TestVL

Attributs

char text[512]

bool creator
bool paused
bool granted
char federateName[512]

RTIfedTime localTime
RTIfedTime TIME_STEP
ObjectClassHandle ObjectClassID
AttributeHandle AttributeID
InteractionClassHandle InteractionClassID
ParameterHandle ParameterID
ObjectHandle ObjectID

Méthodes

void step()

void setCreator(bool)
bool getCreator()
void setPaused(bool)
bool getPaused()
void setFederateName(char*)
char* getFederateName()
void setText(char*)
char* getText()
void setGranted(bool)
void getGranted()

void createFederation()
void joinFederation()
void getHandles()
void regulatedTime()
void publishAndSubscribe()
void sendAttributes()
void sendAttributesWithTime (RTIfedTime)
void actualiserTemps()

void reflectAttributeValues(ObjectHandle, const AttributeHandleValuePairSet&,
const FedTime&, const char*)
void reflectAttributeValues(ObjectHandle,const AttributeHandleValuePairSet&,
const char *theTag)
void federationSynchronized(const char *label)
void timeAdvanceGrant(const FedTime&)
void discoverObjectInstance(ObjectHandle, ObjectClassHandle, const char *)

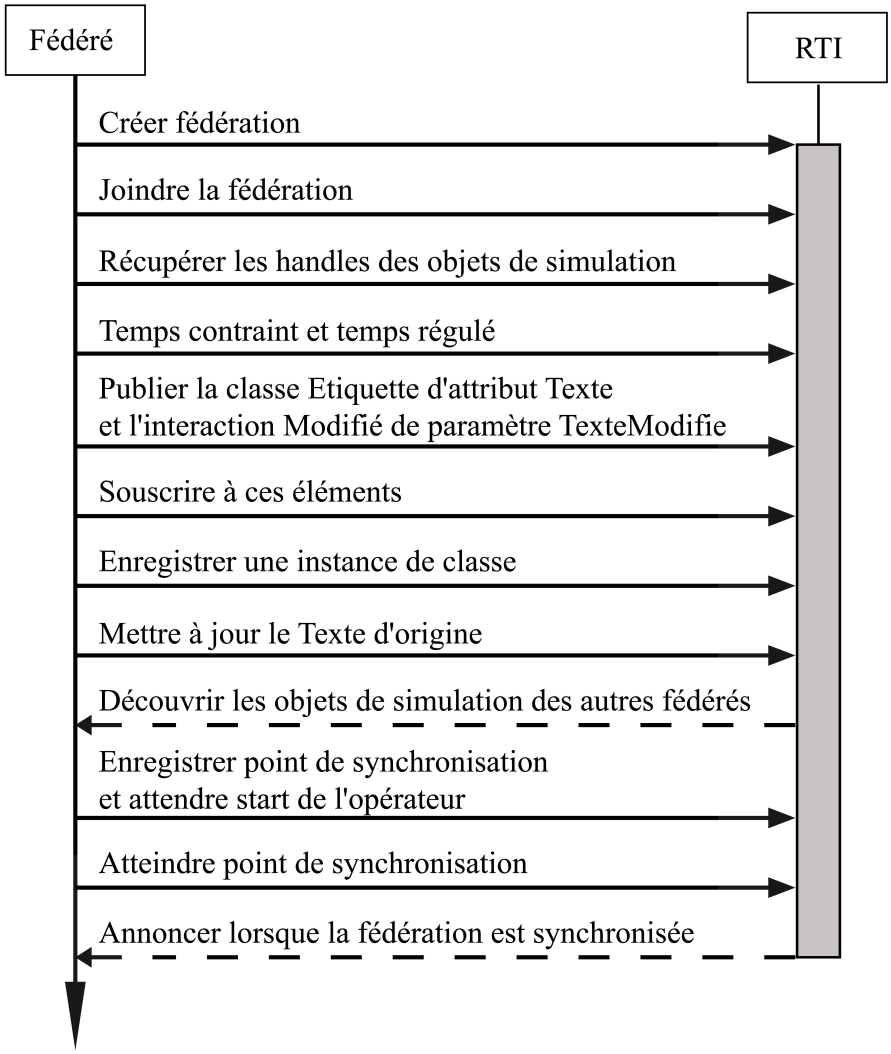
2.2 Le cycle de vie de la fédération TestVL

Le cycle de vie d'une application typique HLA se divise en trois phases. Une phase d'initialisation qui prépare la fédération, la phase de simulation pendant laquelle les attributs des fédérés évoluent et interagissent et enfin une phase de fin de simulation qui détruit la les instances et la fédération.

2.2.1 Initialisation de la simulation

L'initialisation doit préparer la fédération, c'est à dire arriver à un état ou plusieurs fédéré sont capable d'échanger leurs attributs avec le RTI et sont près à démarrer la simulation de manière synchroniser. Pour la synchronisation de notre programme nous avons voulu nous calquer sur ce qui a été fait dans le billard dans lequel après avoir créer la fédération le premier fédéré déclare un point de synchronisation puis se met en pause pour attendre que d'autres fédérés joignent la fédération. Tant que le créateur n'a pas rejoint un point les autres l'attendent au point de synchronisation. C'est donc le créateur qui va lancer la fédération, une fois que tout le monde est près et synchronisé.

Le diagramme UML de cette étape pour le créateur de la fédération de notre application est présenté ci-dessous :



Les autres fédérés suivent les mêmes étapes à l'exception de la création de la fédération qu'ils ne font pas et l'étape de synchronisation qui est différente. Eux atteignent directement le point de

synchronisation et attendent le callback du RTI qui annonce que tous les fédérés l'ont atteint. Ensuite seulement la fédération peu commencer à évoluer.

2.2.2 La boucle de Simulation

Pendant la simulation, c'est l'utilisateur qui va à chaque pas de temps changé le texte des fédérés. La simulation va récupérer tout ces textes et les afficher. C'est aussi l'utilisateur qui va annoncer la sortie de cette boucle et la fin de la simulation. L'algorithme de cette boucle est présenté ci-dessous.

```
Fini=0
Tant que Fini=0(
     Pour i de 0 à 3 (
          Mettre l'attribut granted du fédéré à False
          localTime= temps actuel
                         //queryFerateTime()
          TimeAux=localTime + timeStep
          Demande au RTI d'un avance de temps au temps Time Aux
                         //timeAdavanceRequest()
          Tant que granted=False(
               Faire tick()
               )
                         //Attendre que l'avance de temps soit
                         //accordée l'implémentation du
                         //callback timeAdvanceGranted fait passer
                         //granted à True
          Mettre l'attribut granted du fédéré à False
          Changer l'état de l'attribut text (utilisateur)
          Mettre à jour les attributs de la fédération avec
               l'estampille   localTime+timeStep
          Récupérer les attributs des autres fédérés et les
               afficher.
                         //Callback ReflectAttributeValue avec
                         //marqueur temporel
                         //son implémentation doit permettre
                         //d'afficher les attributs text des autres
                         //fédérés
     )
     Mettre Fini à 1 pour sortir de la boucle (utilisateur)
)
```

Algorithme de la boucle de simulation

2.2.3 La fin de la simulation

La fin de la simulation permet de détruire tout ce qui a été créée pendant l'initialisation. Le fédéré « classique », détruit son instance de classe et quitte la fédération. En plus de ces deux étapes, le créateur détruit la fédération.

3 Plugin HLA pour Flight Gear

3.1 Flight gear et son plugin

3.1.1 Flight Gear

Flight Gear est un simulateur de vol sous licence GNU GPL, dont le développement a débuté en 1996. Il est principalement écrit en C++, et sa licence permet d'en modifier le code source.

3.1.2 Rôle du plugin existant

Nous nous sommes appuyés sur un plugin existant qui gère HLA pour Flight Gear. Il s'intègre dans un ensemble plus grand appellé VirtualAir, qui regroupe différentes ressources permettant de visualiser sur une machine la position d'avions simulés sous différents simulateur et sur différentes machines.

En conséquence le rôle de Virtual Air n'est pas de faire du vol en patrouille, mais de pouvoir visualiser sur une machine tierce les évolutions d'avions. C'est pourquoi ce plugin ne gère que la publication des attributs de l'avion, mais ne prends pas en compte les autres avions.

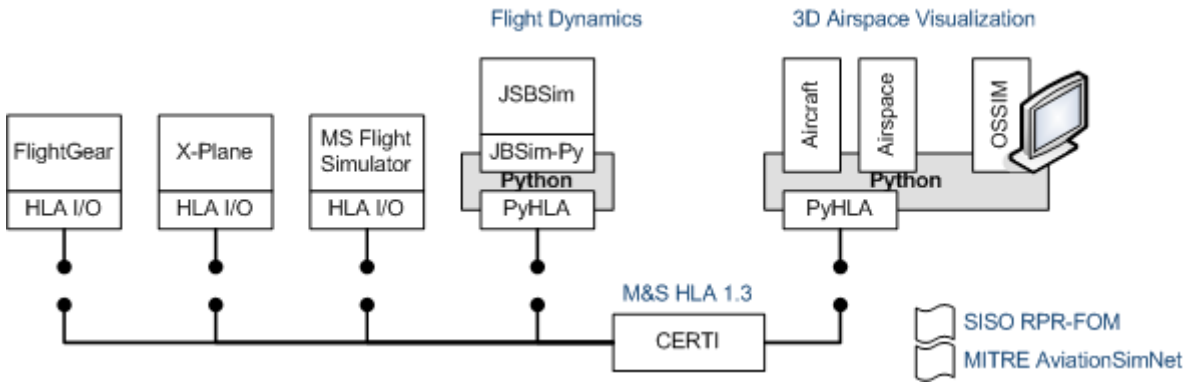


Illustration 1: Structure de l'ensemble Virtual Air

3.1.3 Architecture du plugin existant

Le plugin se présente sous la forme d'un fichier .patch, qui modifie certaines lignes des sources de FlightGear afin de permettre l'implantation du module. Le module se compose d'un fichier .hh et d'un fichier .cc représentant la classe FGHLA (pour Flight Gear HLA). La classe FGHLA comporte des méthodes communes aux modules de mise en réseau, qui sont appelées par Flight Gear.

Étudions le prototype simplifié de la classe FGHLA qui est la colonne vertébrale du plugin :

```
class FGHLA :  public FGProtocol, public NullFederateAmbassador
{
  public:
    FGHLA(...);
    virtual ~FGHLA()

    void discoverObjectInstance(...);
    void reflectAttributes(...);

//FGProtocol methods
    bool open();
    bool close();
    bool process();

//HLA Callbacks
    void reflectAttributeValues(...);
    void removeObjectInstance(...);
    void removeObjectInstance(...);

  private:
    RTI::RTIambassador mRtiAmbassador;

//Maps HLA objects to the FGAIMultiplayer
    typedef std::map<
        RTI::ObjectHandle, osg::ref_ptr<FGAIMultiplayer> >
        MultiPlayerMap;
    MultiPlayerMap mMultiPlayerMap;
};
```

Nous remarquons que la classe FGHLA hérite de deux classes et leurs méthodes :

- NullFederateAmbassador, dont les méthodes sont appellés Callbacks, et permettent de définir l'action du plugin lorsqu'il reçoit des messages venant de la fédération via la RTI.
- FGProtocol, qui dispose de méthodes dédiées à la mise en réseau sous FlightGear, et permet à FlightGear de dialoguer avec le module.

De plus, cette classe dispose de deux attributs importants :

- RTIambassador, qui est implanté conformément à ce que nous avons exposé plus haut, en tant qu'attribut du fédéré, cette attribut permet au plugin d'envoyer des messages de mises à jour à la fédération, via la RTI.
- MultiPlayerMap, qui permet au module de faire le lien entre les objets internes à FlightGear, et les Objets de la fédération.

Il est clair dans son architecture que la classe FGHLA joue le rôle d'interface entre FlightGear d'un côté, et HLA de l'autre.

3.2 Modification du plugin

3.2.1 Objectif

L'objectif de la modification est de réaliser un plugin qui permette de visualiser dans chaque instance de Flight Gear les autres avions simulés par d'autres fédérés, afin de pouvoir réaliser un vol en patrouille. En conséquence le plugin doit publier les attributs de l'avion au sein de la fédération, et souscrire aux attributs des autres avions de la simulation afin de pouvoir les afficher.

3.2.2 Mise en œuvre

Pour modifier le comportement de la simulation, le plus est de modifier les services rendu par les méthodes open(), close() et process().

Pour pouvoir arriver à faire du vol en patrouille la première étape est de préparer la simulation. Il faut donc tout d'abord que plusieurs avions puissent joindre la fédération et être synchronisés. On pourra pour cela reprendre l'étape de synchronisation qui a été faites dans l'application Essai1 pour l'adapter et insérer ce service dans la méthode open() du plugin.

Au niveau de la gestion du temps dans le plugin est à mettre en place. On pourra aussi se calquer sur ce qui à été fait dans notre application, sur le régime de temps régulé et contraint, sur l'implémentation du callback timeAdvanceGranted et sur la gestion du temps mis en place dans la boucle de simulation.

Il faudra en outre définir des interaction propre au vol en patrouille, les insérer dans le .fed, en implémenter les méthodes et les prendre en compte dans la fonction process() du plugin.

Cependant d'autres questions se posent pour pouvoir arriver au vol en patrouille comme par exemple celle du plan de vol. Comment initialiser un plan de vol avec Flight Gear? Quelles interactions est-il judicieux de définir? Nos succeurs devront y répondre.

Afin d'aborder les difficultés les unes après les autres, nous pensons qu'en transformant un peu notre application et en ajoutant la synchronisation à la fonction open(), il est possible de créer une fédération avec un Flight Gear réel et un autre fédéré qui simule au point de vu de la fédéation un autre Flight Gear. Cet autre fédéré pourrait par exemple récupérer les attributs du Flight Gear et les afficher. Ceci constituerai une première étape de modification du plugin.

Conclusion

Si nous n'avons pas eu le temps d'aboutir à des résultats concluant dans la simulation de Flight Gear, nous fournissons grâce à notre application et à ce document, les outils pour une compréhension rapide du fonctionnement d'HLA, de l'implémentation d'un fédéré, des services rendu par le plugin HLA-Flight Gear et de la marche à suivre pour poursuivre notre travail. En effet nous avons tenter de fournir les outils qui nous auraient permis d'avancer plus vite comme par exemple l'annexe 1 de documentation des classes de CERTI.

Annexe 1 : Documentation des principales méthodes de la classe RTIambassador utilisées dans la classe TestVL

void createFederationExecution(char* NOM_FEDERATION,char* Nom_Fichier.fed)
Créer la fédération nommée Nom_FEDERATION dont les fédérés sont décrit dans le fichier Nom_Fichier.fed.
Elle renvoie une exception si la fédération existe. Cette exception est utilisée pour distinguer le créateur des autres fédérés.

void joinFederationExecution(char* Nom_Fédéré,char NOM_FEDERATION,classe Classe_Du_Fédéré)
Permet de joindre le fédéré avec comme nom Nom_Fédéré, à la fédération nommée NOM FEDERATION.

void enableTimeRegulation(RTIfedTime localTime,RTIfedTime TIME_STEP)
Permet de passer le fédéré en mode temps régulé.

void enableTimeConstrained()
Active le régime de temps contraint

ObjectClassHandle getObjectClassHandle(char* Nom_Classe)
Renvoie le handle associé à la classe Nom_Classe. Celui-ci est unique pour toute la fédération

AttributeHandle getAttributeHandle(char* Nom_Attribute,ObjectClassHandle ObjectClassHandle)
Renvoie le handle de l'attribut Nom_Attribute de la classe qui à pour handle ObjectClassHandle

InteractionClassHandle getInteractionClassHandle(char* Nom_Interaction)
Renvoie le handle de l'interaction Nom_interaction.

ParameterHandle getParameterHandle(char* Nom_Parametre ,InteractionHandle InteractionClassHandle)
Renvoie le handle du paramétre Nom_Parametre associé à l'interaction de handle InteractionClassHandle.

void add(RTI::AttributeHandle AttributeID)
Permet d'ajouter l'attribut de handle AttributeID dans un AttributeHandleSet

void publishObjectClass(RTI::ObjectClassHandle ObjectClassID, AttributeHandleSet *attributes)
Permet de publier les attributs de la classe de handle ObjectClassHandle qui sont contenue dans l'AttributeHandleSet* attributes

void subscribeObjectClassAttributes(RTI::ObjectClassHandle ObjectClassID, RTI::AttributeHandleSet *attributes, RTI::RTI_TRUE)
Permet au fédéré de s'abonner à ces attributs.

void registerObjectInstance(RTI::ObjectClassHandle ObjectClassID,char* Nom_Fédéré)

Permet d'enregistrer l'instance de la classe de handle ObjectClassID du fédéré Nom_Fédéré

void registerFederationSynchronizationPoint(char* label, char* Tag)
Permet d'enregistrer un point de synchronistion repéré par le label Label.

void synchronizationPointAchieved(char* Label)
Permet d'atteindre le point de synchronisation repéré par le label Label. Cette méthode déclenche l'envoie d'un message du RTI (callback) utilisant la méthode federationSynchronized qui annonce que le point de synchronization est bien atteind.

void tick()
Permet de laisser le temps au RTI d'exécuter les différentes action demandées par le fédéré.

void requestObjectAttributeValueUpdate(ObjectClassHandle ObjectID,AttributeHandleSet *attributes)
Demande au RTI de mettre à jour les attributs de la classe de handles ObjectID contenu dans attributes.

void updateAttributeValues(ObjectClassHandle ObjectID,AttributeHandleSet *attributeSet, char *MiseAJour)
Mettre à jour ses attributs au niveau de la fédération. Cette méthode déclenche l'envoie d'un message du RTI (callback) utilisant la méthode reflectAttributValue qui renvoie les attributs des autres fédérés.

void updateAttributeValues(ObjectClassHandle ObjectID,AttributeHandleSet *attributeSet, RTIfedTime Time, char *MiseAJour)
Mettre à jour ses attributs au niveau de la fédération et d'aonncer cette mise à jour au temps Time. Cette méthode déclenche l'envoie d'un message du RTI (callback) utilisant la méthode reflectAttributValue qui renvoie les attributs des autres fédérés au temps Time.

void queryFederateTime(RTIfedTime& LocalTime)
Mettre dans LocalTime la valeur de temps actuel du fédéré.

void timeAdvanceRequest(RTIfedTime Time)
Demande au RTI une avance jusqu'au temps Time. Cette méthode déclenche l'envoie d'un message du RTI (callback) utilisant la méthode timeAdvanceGrant qui préviens lorsque l'avance de temps à été accordée.

Void resignFederationExecution(RTI::DELETE_OBJECTS_AND_RELEASE_ATTRIBUTES)
Quitter la fédération.

void destroyFederationExecution(char* NOM_FEDERATION)
Permet de détruire la fédération nommée NOM_FEDERATION.

Annexe 2 : Fichiers sources : TestVL.cc, TestVL.hh Essai1.cc et TestVL.fed

Bibliographie

Rapports de stage :

Katerine Quince :
Simulation distribuée HLA scientique sur clusters

Patrick Desseaux :
Prototypage d'un système de simulation distribuée compatible avec HLA

Ouvrages :

Dr. F. Kuhl, Dr. R. Weatherly, Dr. J. Dahmann
Creating simulation systems

Documentation fournie avec Flight Gear

Thèse :

J. Latour
DARE-HLA, un intergiciel pour la conception et la réutilisation des fédérés et fédérations HLA

Sites internet :

http://www.cert.fr/CERTI
http://www.cplusplus.com