Utilisation d’une approche orientée automates pour la refonte de l’API de communication de Coloane

PSAR

Rapport mi-parcours et éléments de conception

Etudiants : Kahina BOUARAB
Youcef BELATTAF

TABLE DES MATIERES

1. INTRODUCTION

L'équipe MoVe du LIP6 a développé une plateforme logicielle de modélisation et de vérification de systèmes d'information appelée FrameKit. Cette plateforme est utilisée par un nombre important d'équipes de recherche dans le monde.

Pour compléter cette plateforme, deux clients graphiques ont été développés, Macao et Coloane. Le premier, historique, est le client de référence, fonctionne sous MacOS et SunOS. Le second quant à lui est récent (2006) et est multi-plateforme, cependant, il n'est pas encore stable et des fonctionnalités restent à implémenter.

Coloane se compose de quatre modules : l'éditeur graphique, le moteur de formalismes, le module de communications et les interfaces. Il sera question ici de la refonte du module de communication avec FrameKit.

Le protocole de communication entre Coloane et FrameKit est le CAMI. Il s'agit d'un protocole de description textuelle. Il permet de décrire les modèles, les menus de service, les objets à afficher tels que les boites de dialogue, les aspects liés à la gestion des sessions ...

Notre travail consiste à concevoir une nouvelle architecture et réaliser le module de communication (un API de communication) de Coloane.

Ce rapport constitue une analyse et une première proposition d'architecture de cet API. Nous allons commencer par l'étude du protocole CAMI. Ensuite nous présenterons des éléments d'architecture.

2. Description des phases d'échanges avec FrameKit

Dans cette section, il s'agit de présenter le protocole CAMI. Nous avons choisi de le faire en utilisant les diagrammes de séquence et des automates. Ceci dans illustrer les échanges (commandes textuelles) lors des différentes phases de communication entre Coloane et FrameKit.

2. 1. Phase d'authentification

Diagramme de séquence

Le diagramme de séquence suivant illustre les échanges opérés entre COM, API et FrameKit lors de la phase de l'authentification.

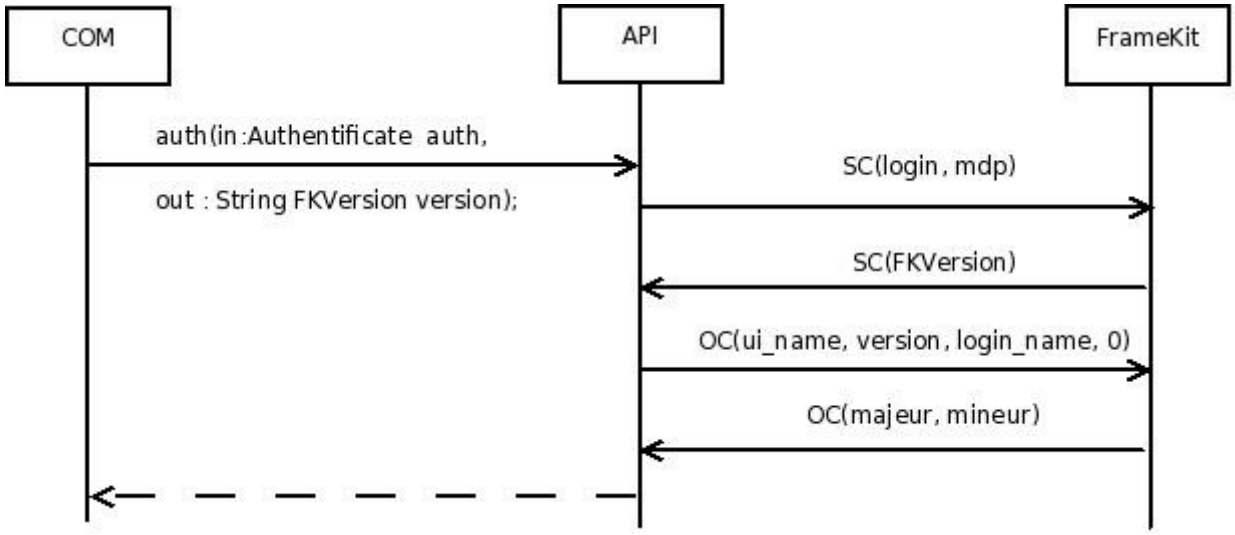

Figure 1: Diagramme de séquence de l'authentification

Description des informations échangées

Informations en entrée (input):
- login : nom d'utilisateur.
- mdp : mot de passe.
- ip : Adresse IP où réside FrameKit.
- port : Port sur lequel FrameKit écoute.
- ui_name : nom du client (Coloane)[1].

Information en sortie (output)
- FKVersion : version de FrameKit.
- majeur : numéro de version majeur.
- mineur : numéro de version mineur.

1 Ce paramètre est connu lors de l'initialisation de l'API, il n'est donc pas passé à la méthode auth.

Automate des échanges avec FK (automate du protocole cami)

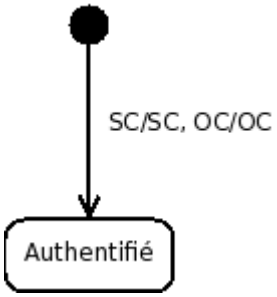

Figure 2 : Automate de l'authentification.

Format des objets

Classe Autheticate{
login : String;
mdp : String;
ip: String ;
port: int ;
}

Classe FKVersion{
fkVersion : String;
majeur : int;
mineur : int;
}

2. 2. phase ouverture d'une session

Cette phase est décrite en trois parties :
- Demande de connexion d'un modèle à FrameKit.
- Réception des questions (menus).
- Réception des modifications sur ce menu.

2. 2. 1. Connexion du modèle

Diagramme de séquence

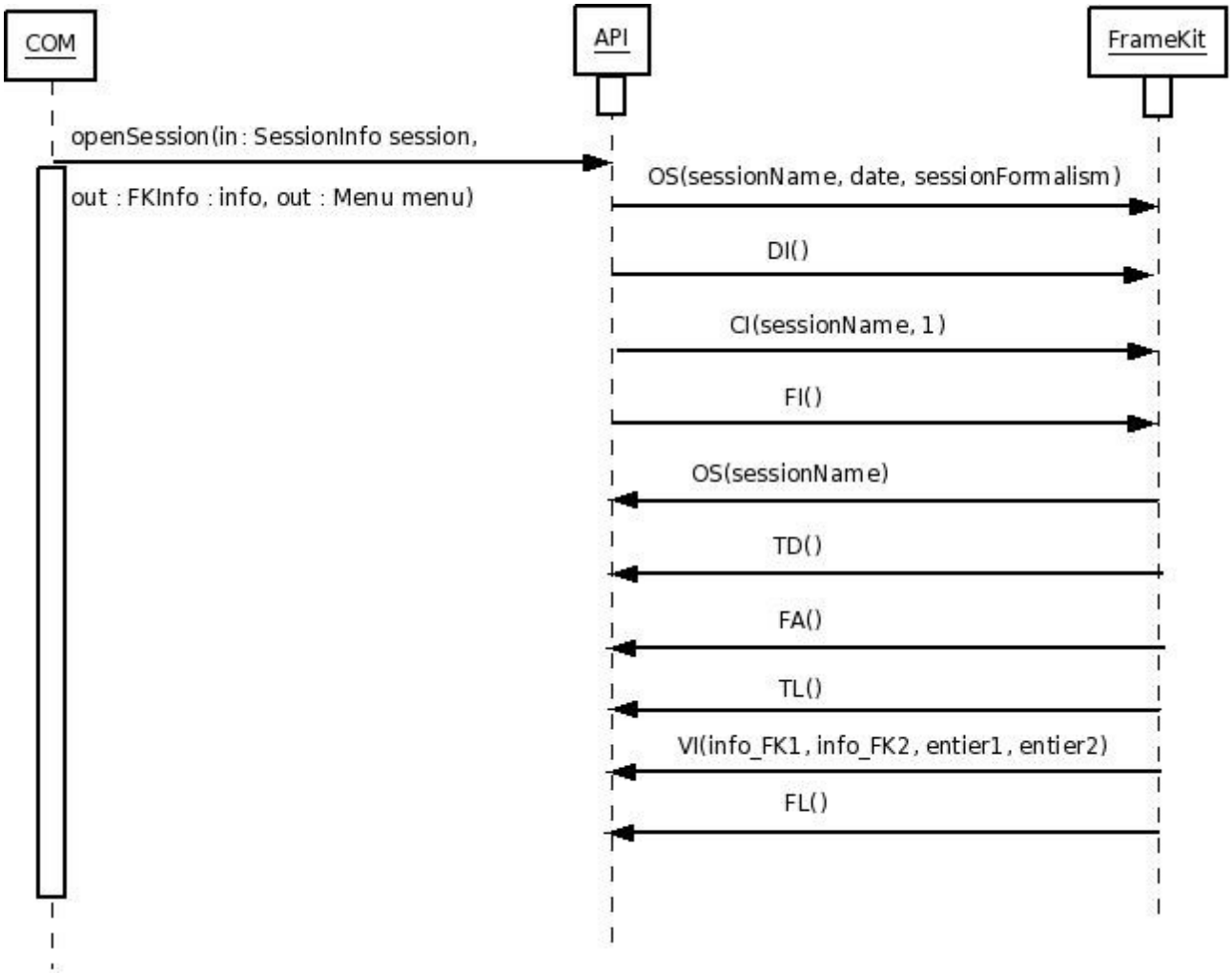

Figure 3: Diagramme de séquence de la demande d'ouverture d'une session.

Description des informations échangées

Informations en entrée (input) :

- sessionName : nom de la session (nom du modèle).
- date : date du modèle, affectée lors de l'ouverture de la session.
- sessionFormalism : formalisme du modèle (Ex : CPN-AMI).

Informations en sortie (output) :

- info_FK1 : « FrameKit environment»
- info_FK2 : « FrameKit Environment, by F.Kordon, LIP6 »
- Incremental
- RésultatCalculé

Format des objets

Classe SessionInfo{
    sessionName : String;
    sessionDate : int;
    sessionFormalism : String;
}

Classe FKInfo{
    FKInfo1 : String;
    FKInfo2 : String;
    entier1 : int;
    entier2: int;
}

Automate Cami de la phase de connexion du modèle

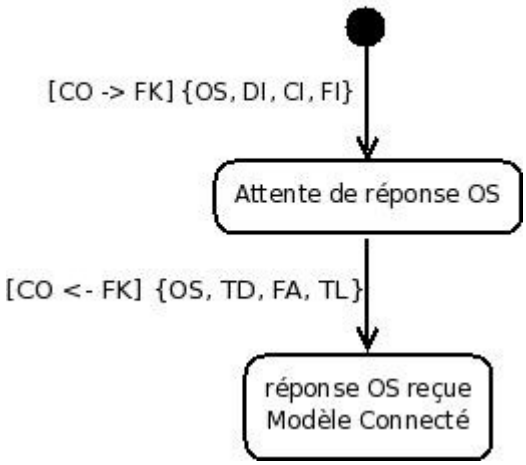

Figure 4: Automate Cami de la phase de connexion du modèle

2. 2. 2. Réception des questions

Diagramme de séquence

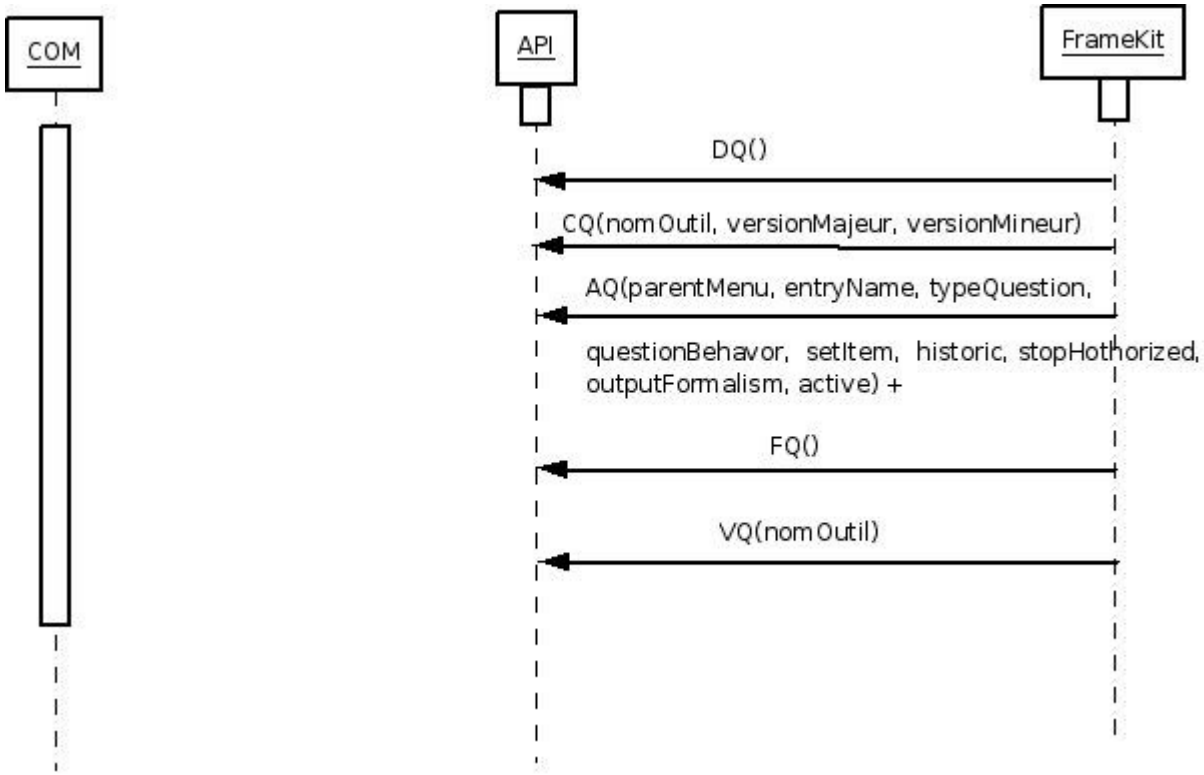

Figure 5 : Diagramme de séquence de la phase de réception des questions.

Description des informations échangées

- Le DQ indique le début de transmission d'un arbre de service.
- Le CQ demande la création d'une arborescence et passe les paramètres suivants :
  - nomOutil : nom de l'outil correspondant au menu de service (CPN-AMI).
  - versionMajeur, versionMineur : version de CPN-APN.

- Le AQ ajoute une question dans l'arbre de service.
  - parentMenu : menu parent de la question (père).
  - entryName : nom du service (fils).
  - questionType : type de la question.
  - question_behavior :
  - setItem
  - historic
  - stopAuthorized : le service peut-il être interrompu par l'utilisateur.
  - outputFormalism
  - active : la question est-elle active.

- Le FQ indique la fin de trasmission d'un menu de service.
- Le VQ demande l'affichage du menu principal.

Format des objets

Le retour de la méthode openSession se fera donc à la suite de la réception d'un QQ(3). Ainsi au retour de cette méthode COM dispose du menu de service qui est l'objet Menu . Ce dernier est construit au fur et à mesure de la réception des AQ par API. Cette partie sera détaillée dans la section 5.

Automate Cami de la phase de réception des menus

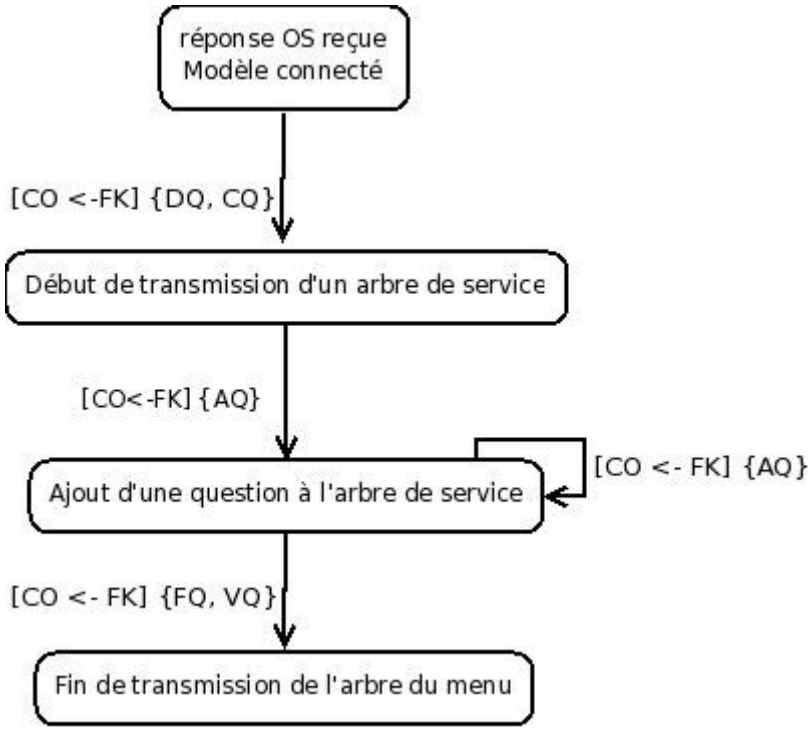

Figure 6 : Automate Cami de la phase de réception des questions.

2. 2. 3. Modification des menus

Diagramme de séquence

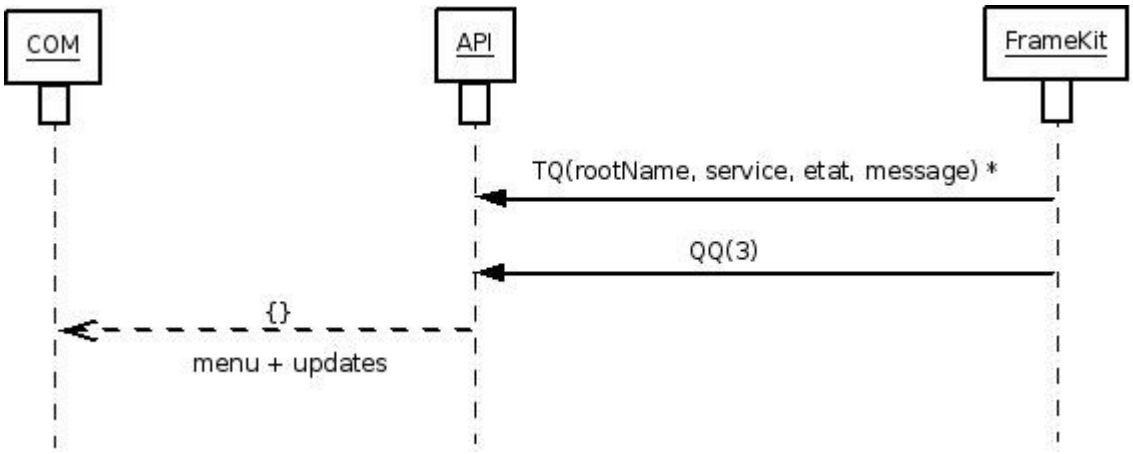

Figure 7: Diagramme de séquence de la phase de modification des menus.

Description des informations échangées

- le TQ est un modificateur de l'arbre de service.
  - rootName : nom du service principal.
  - service : nom du service.
  - etat : état du service :
    - 2 : message d'information .
    - 7 : activer la question
    - 8: désactiver la question.
  - message : Chaine de caractère à passer à coloane.
- le QQ(3) est envoyé à la fin des TQ.

Automate Cami de la phase de modification des menus

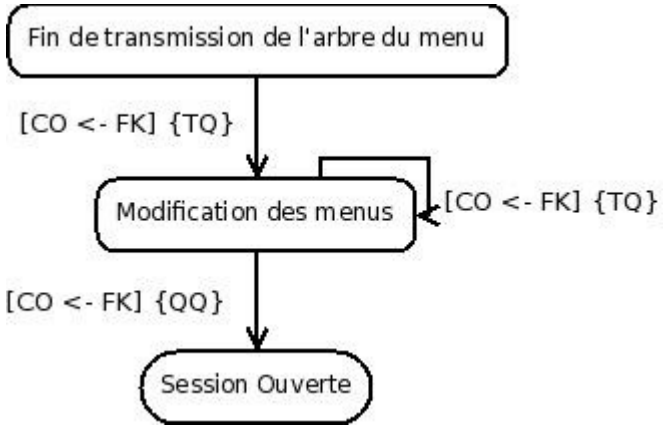

Figure 8 : Automate Cami de la phase de modification des menus

Format des objets

Les modifications (TQ de type 7 et 8)sont envoyées avec le menu de service, donc au retour de l'ouverture de session. Évidement, les TQ(2) qui sont des messages à envoyer en temps réel à Coloane sont traités immédiatement. Le pattern Observer s'avère ici très intéressant, car cela permettra de notifier COM sans pour autant appeler explicitement COM. Nous parlerons plus de cela dans la partie conception.

2. 3. phase de demande de service

2. 3. 1. Gros plan sur la demande de service

La phase de demande de service est composée de quatre parties, à savoir :
- initiation de la demande de service à FK.
- Demande du modèle de la part de FK.
- Envoi du modèle à FK.
- Réponse à la demande de service.

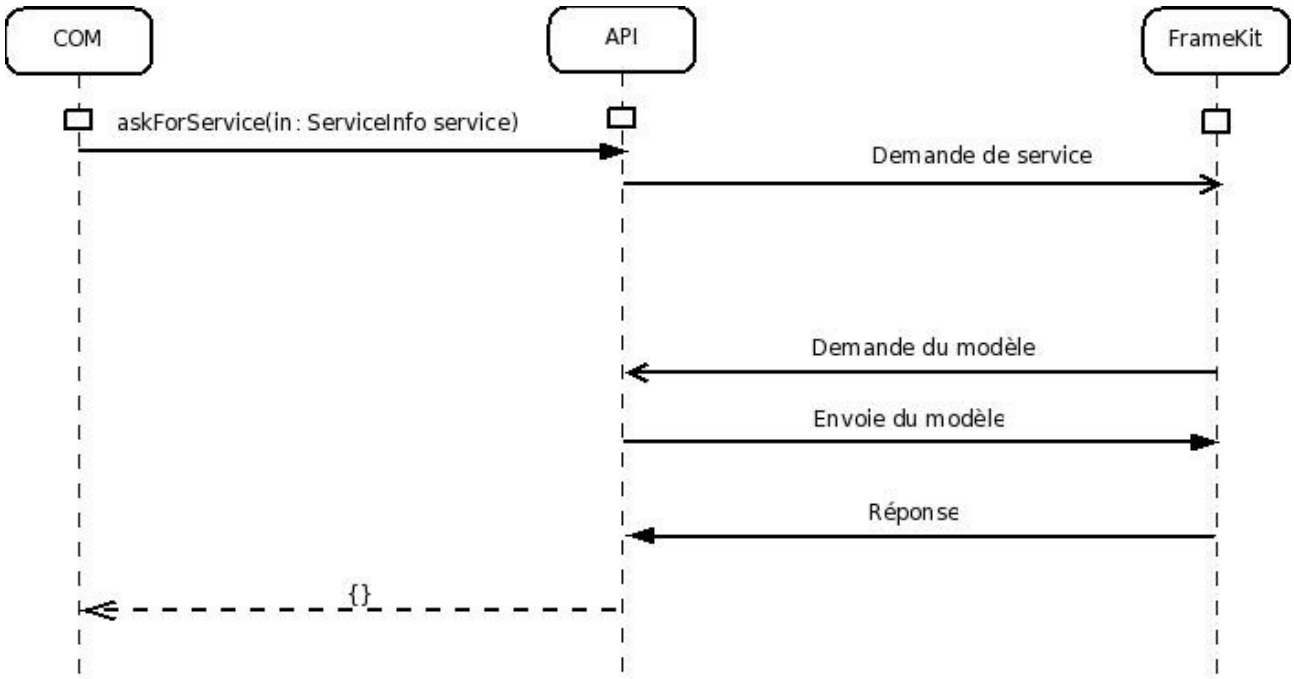

Figure 9 : Gros plan sur la demande de service.

Remarque
La demande et l'envoi de modèle n'apparaissent que dans le cas où la plateforme n'a pas préalablement reçue le modèle (première demande de service ou demande de service après une invalidation du modèle ou modification du modèle).

2. 3. 2. Initiation de la demande de service

Diagramme de séquence

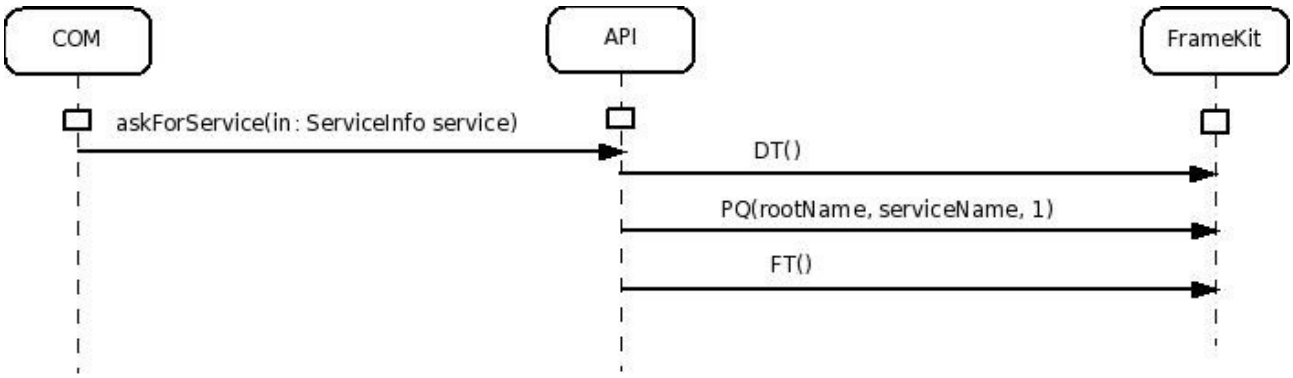

Figure 10 : Diagramme de séquence demande de service.

Description des informations échangées

- DT() sert à annoncer à FrameKit une demande de service.
- PQ transmet la question à FrameKit:
  - rootName : nom du service principal (ex : Ami-net).
  - menuName : sous menu de rootName.
  - serviceName : le service proprement dit.
- FT() indique la fin de la demande.

Format des objets

```
Classe ServiceInfo{
        rootMenu : String;
        menuName : String;
        serviceName : String
}
```

Automate CAMI

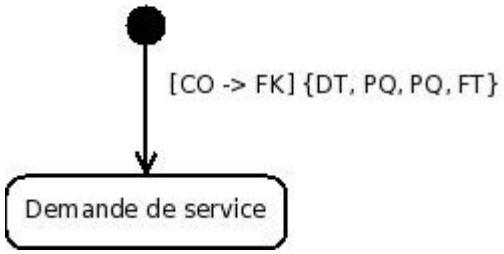

Figure 11 **:** Automate CAMI demande de service

2. 3. 3. Demande du modèle

Diagramme de séquence

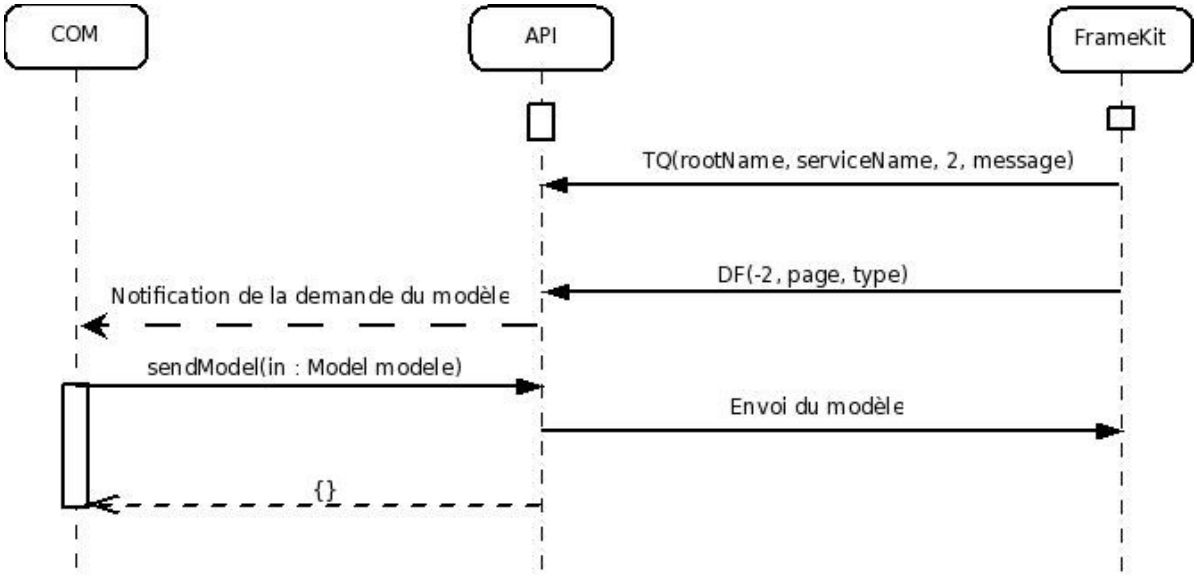

Figure 12 : Diagramme de séquence demande de modèle.

Description des informations échangées

- le TQ informe API que l'outil va demander le modèle (TQ déja décrit dans l'ouverture de session).
- DF demande le modèle à Coloane.
  - -2.
  - pageNumber.
  - TypeQuery.
- L'envoi du modèle : ensemble de commandes encadrées entre DB et FB.

Format des objets

Nous avons parlé du traitement des TQ(2) dans l'étape de l'ouverture de session et de l'utilisation du pattern observer. Nous allons faire de même pour la demande de transmission du modèle, et utiliser le pattern observer.

On définit une interface IModel décrivant le modèle. Cette interface devra être implémentée par une classe Model définie au niveau de COM. Ce qui permettra à API véhiculer le modèle vers FrameKit.
Les méthodes de l'interface Imodel nous permettent d'accéder aux éléments d'un modèle.

Automate CAMI

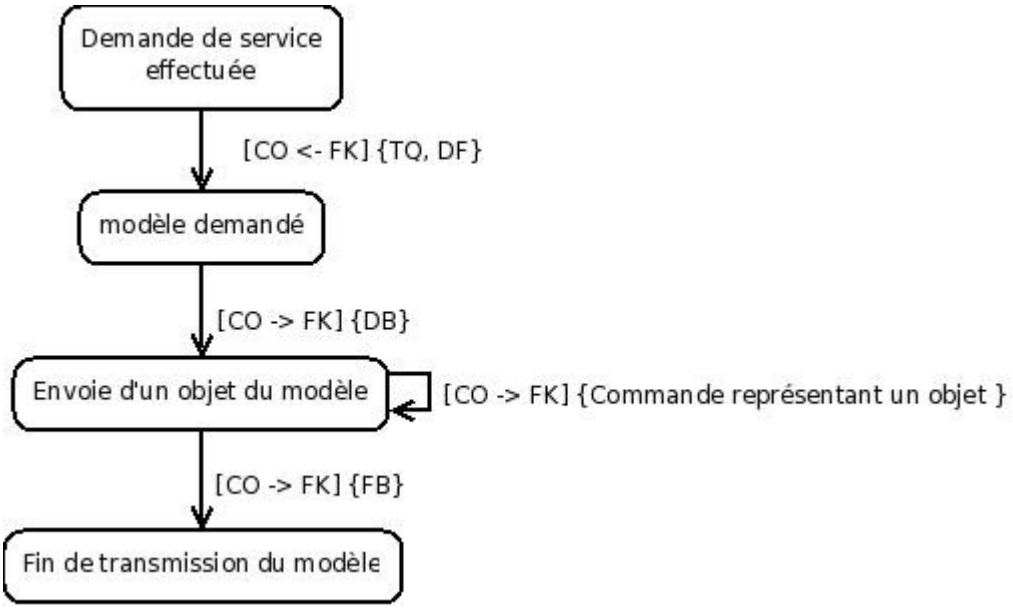

Figure 13 : Automate CAMI - demande du modèle.

2. 3. 4. Réception des réponses

diagramme de séquence

La partie réception des réponses dépend fortement de l'outil appelé. Voici le canevas de la réponse :

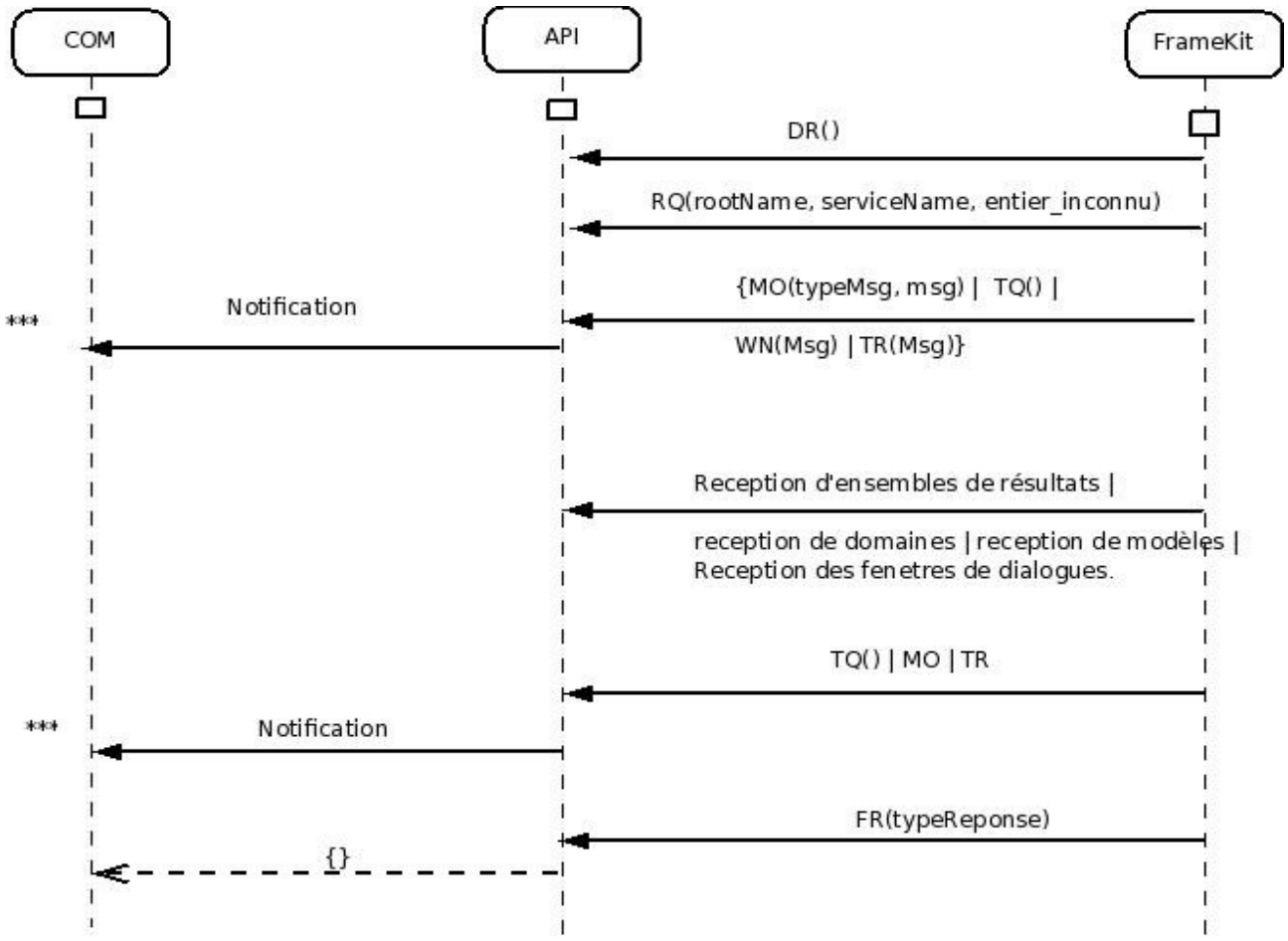

Figure 14 : Canevas réception des réponses

Informations échangées

- DR() : Début de transmission d'une réponse.
- Le RQ désigne la question à laquelle on répond.
  - RootName : nom de la racine correspondant à la question.
  - ServiceName : nom de la question.
  - suiteVaSuivre : Se trouve en troisième argument.

- TR : Message de trace.
  - msg : chaine de caractère du message.

- MO: Message spécial.
  - TypeMsg : type du message (1 à 4).
  - msg : La chaine du message d'avertissement.

- WN : Message d'avertissement.
  - msg : chaine du message d'avertissement.

- FR : Fin de transmission d'une réponse.

- typeReponse : chaine du message d'avertissement (1, 2, 3).

Les messages transmis par les TR sont à écrire dans la fenêtre d'historique, quant à ceux véhiculés par les MO et les WN sont à afficher immédiatement.

Automate CAMI

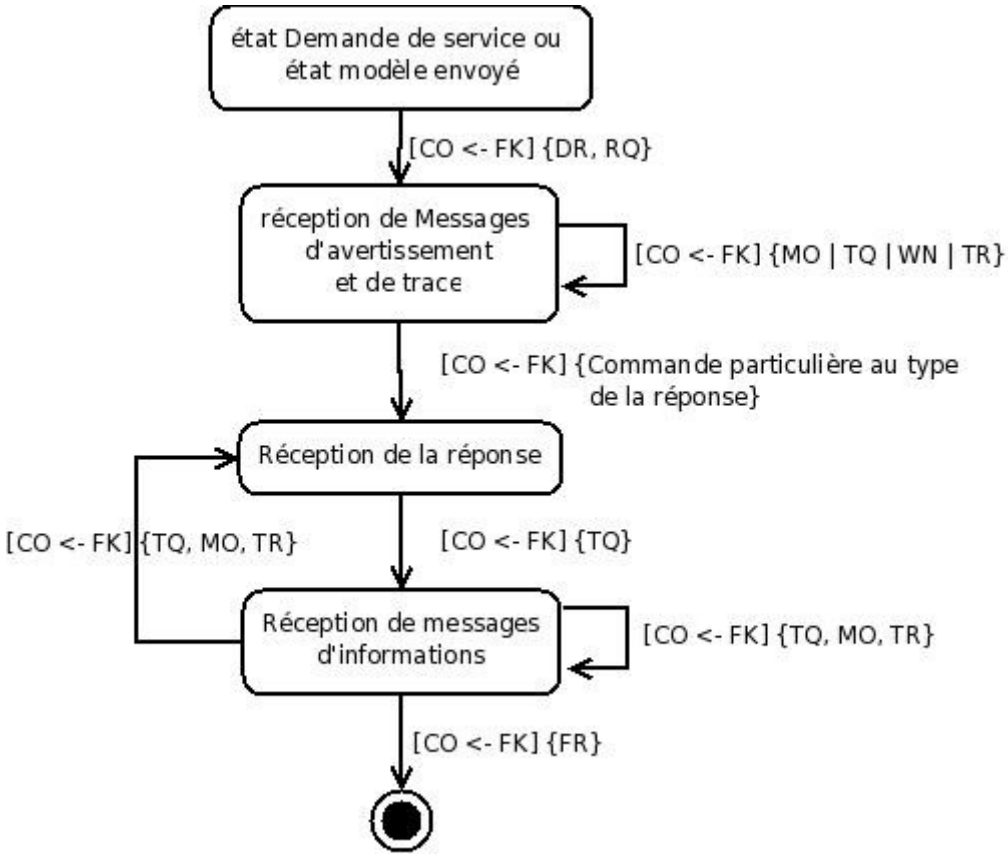

Figure 15 : Automate canevas réception des réponses

2. 3. 4. 1. Réception d'ensembles de résultats

Diagramme de séquence

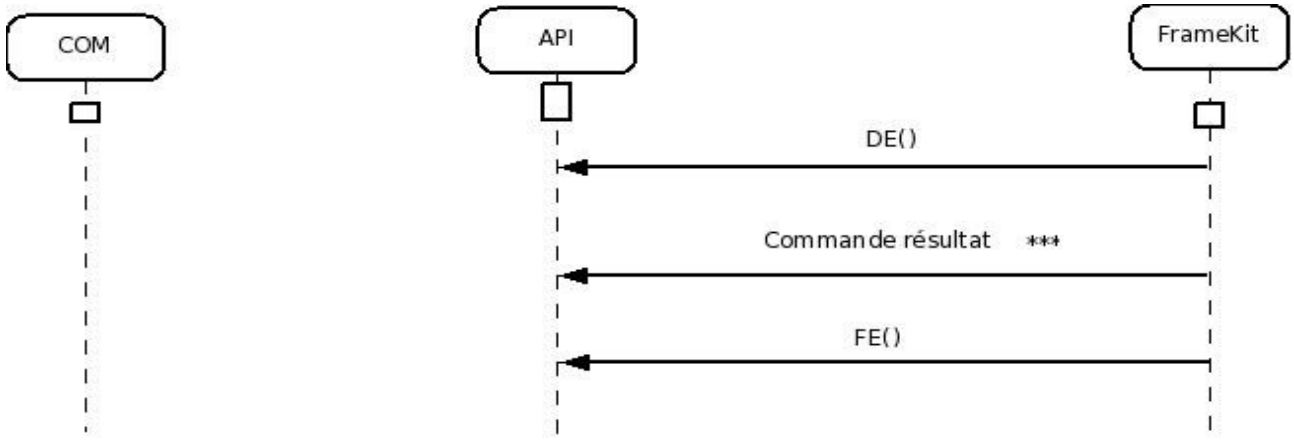

Figure 16 : Diagramme de séquence réception des ensembles de résultats.

Description des informations échangées

- DE : Début d'un ensemble (ou sous ensemble) de résultats associés à une question.
- FE : Fin d'un ensemble (ou sous ensemble) de résultats associés à une question.

Commande de résultat peut être :

- RT (resultText): résultat textuel.
  - resultText : chaine de caractère résultat.

- RO(objectID) : désigne un objet associé au résultat dans un ensemble de résultats.
  - objectID : identificateur d'objet.

- ME(objectID): mise en évidence d'un objet.
  - objectID : identificateur d'objet.

- MT(objectID, typeAttribut, indexDébut, indexFin) : mise en évidence d'un attribut (texte) de l'objet objectID.

- XA(objetID, typeContenu, nouveauContenu) : Remplace la valeur de l'attribut objectID de type typeContenu par une autre valeur.

- SU(objectID) : supprime l'objet objectID du modèle.

- CN(typeNode, numNode) : crée un nouveau nœud numNode de type typeNode.
- CT(typeAttribut, objectID, contenu) : crée un attribut de type typeAttribut pour l'objet objectID.

- CA(typeArc, numArc, numNodeDepart, numNodeStart) : crée un arc numArc de type typeArc, qui relie les nœuds numNodeDepart et numNodeEnd.

Automate CAMI

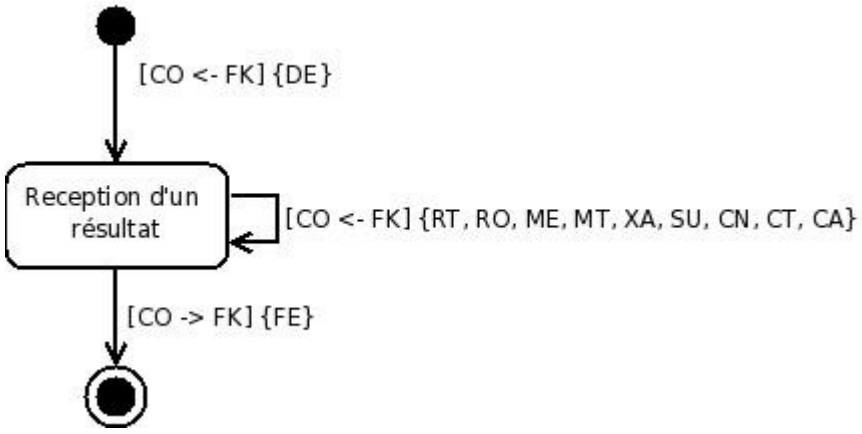

Figure 17 **:** automate réception d'ensembles de résultats.

2. 3. 4. 2. Réception de tables d'attributs

Diagramme de séquence

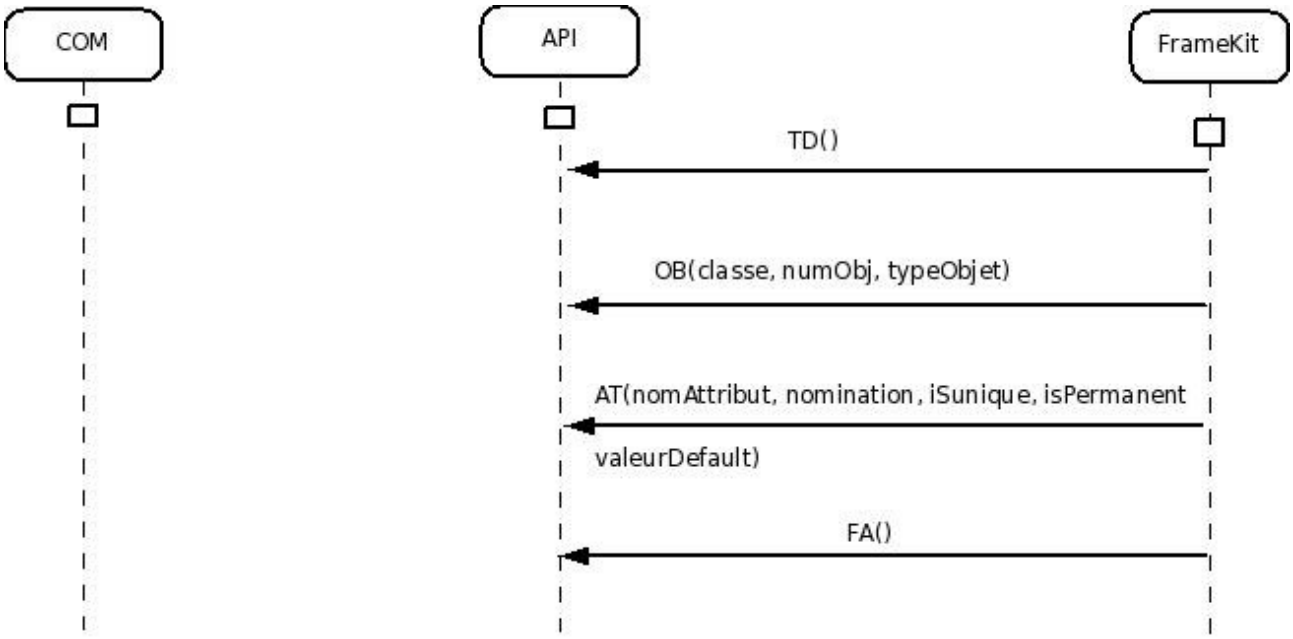

Figure 18 **:** Réception de tables d'attributs.

- TD : Début réception d'une table du domaine.
- OB() : Objet.
  - classe : noeud ou connecteur
  - numObj : numéro de l'objet dans sa classe.
  - typeObj : type de l'objet (ex : place).

- AT() : attribut d'un objet
  - nomAttribut : nom de l'attribut.
  - nomination
  - isUnique : unique ou multiple.
  - isPermanent : permanent ou temporaire pour un service.
  - valeurDefault : valeur par défaut (chaine de caractère)

Automate CAMI

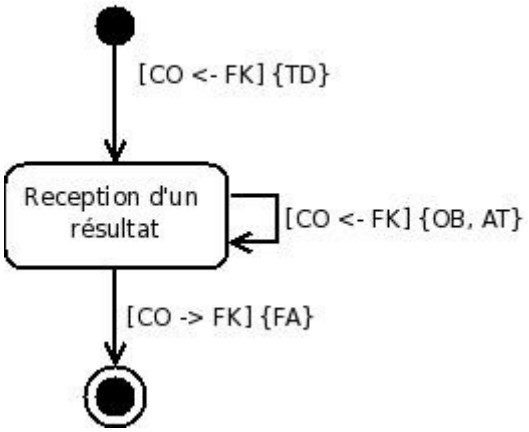

Figure 19 : Automate réception des tables d'attributs.

2. 3. 4. 3. réception de modèles

Diagramme de séquence

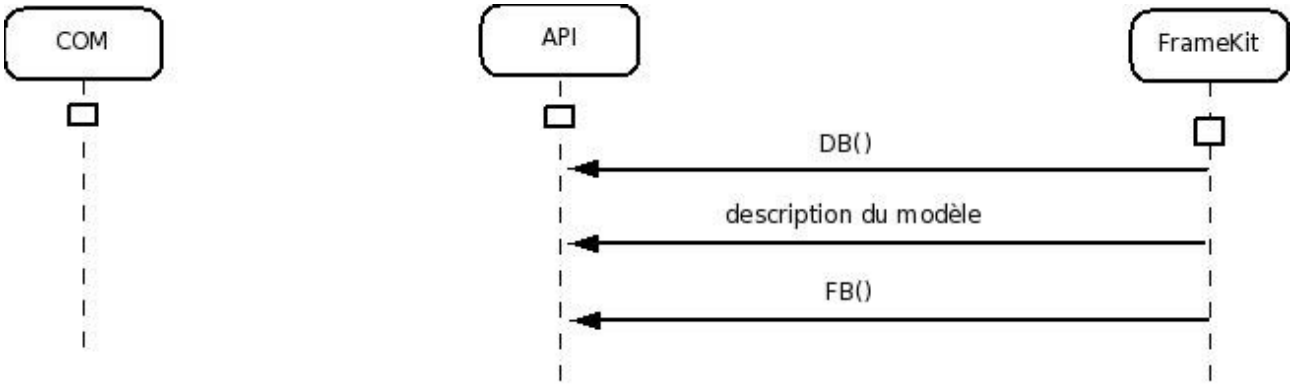

Figure 20 : réception d'un modèle.

- DB() : début de transmission d'un modèle.
- Description du modèle : commandes décrivant un modèle (Pas très utile de les expliciter ici).
- FB() : Fin de transmission de modèle.

2. 3. 4. 4. Réception des fenêtres de dialogue

Diagramme de séquence

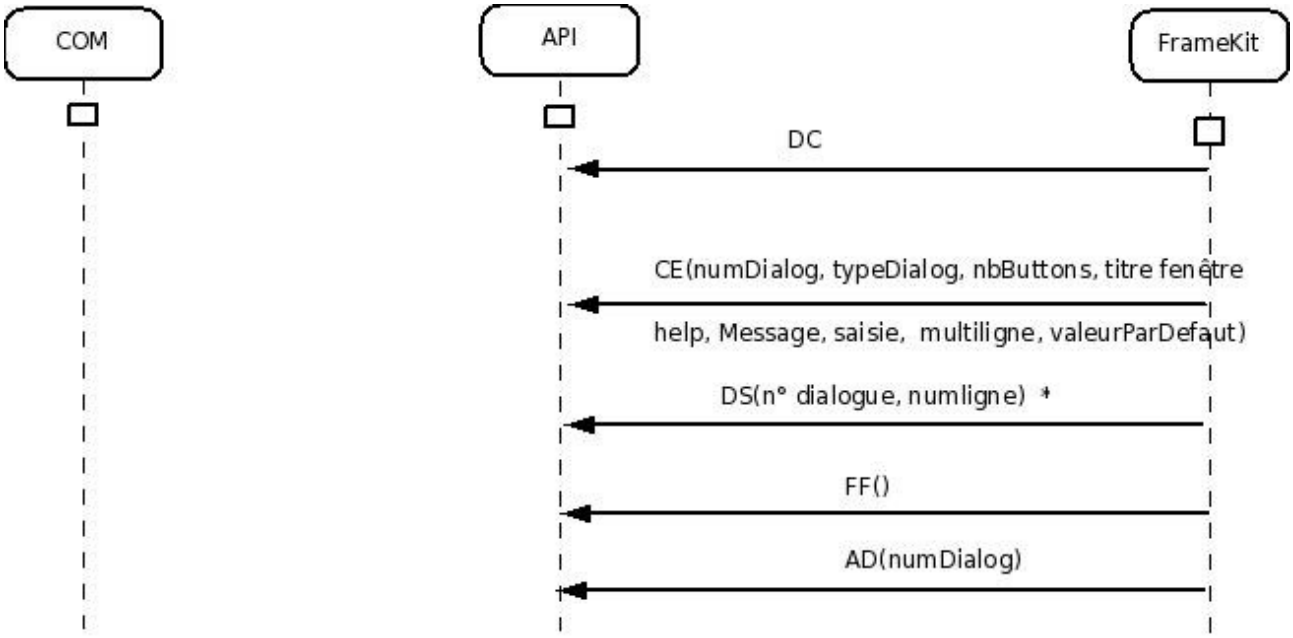

Figure 21 : réception d'une fenêtre de dialogue.

Informations échangées

- DC : début d'un dialogue.
- CE : crée un dialogue.
  - NumDialog : identifiant du dialogue.
  - TypeDialog : type du dialog (standard, warning ...).
  - nbButtons : Nombre de boutons.
  - TitreFenetre : titre de la fenêtre.
  - help
  - message : message de la fenêtre de dialogue.
  - Saisie oui ou non.
  - multiligne : oui ou non.
  - ValeurParDefaut.

- DS : suite du dialogue.
  - NumDialog : numéro du dialog.
  - Numligne : numéro de la ligne.

- FF : Fin d'un dialogue.
- AD : afficher la boite de dialogue.
  - NumDialog : numéro de la boite de dialogue.

Automate Cami

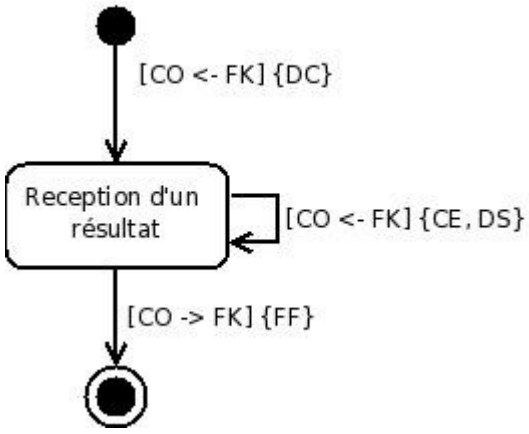

Figure 22 : Automate réception des dialogues.

2. 3. 4. 5. Réponse aux dialogues

Diagramme de séquence

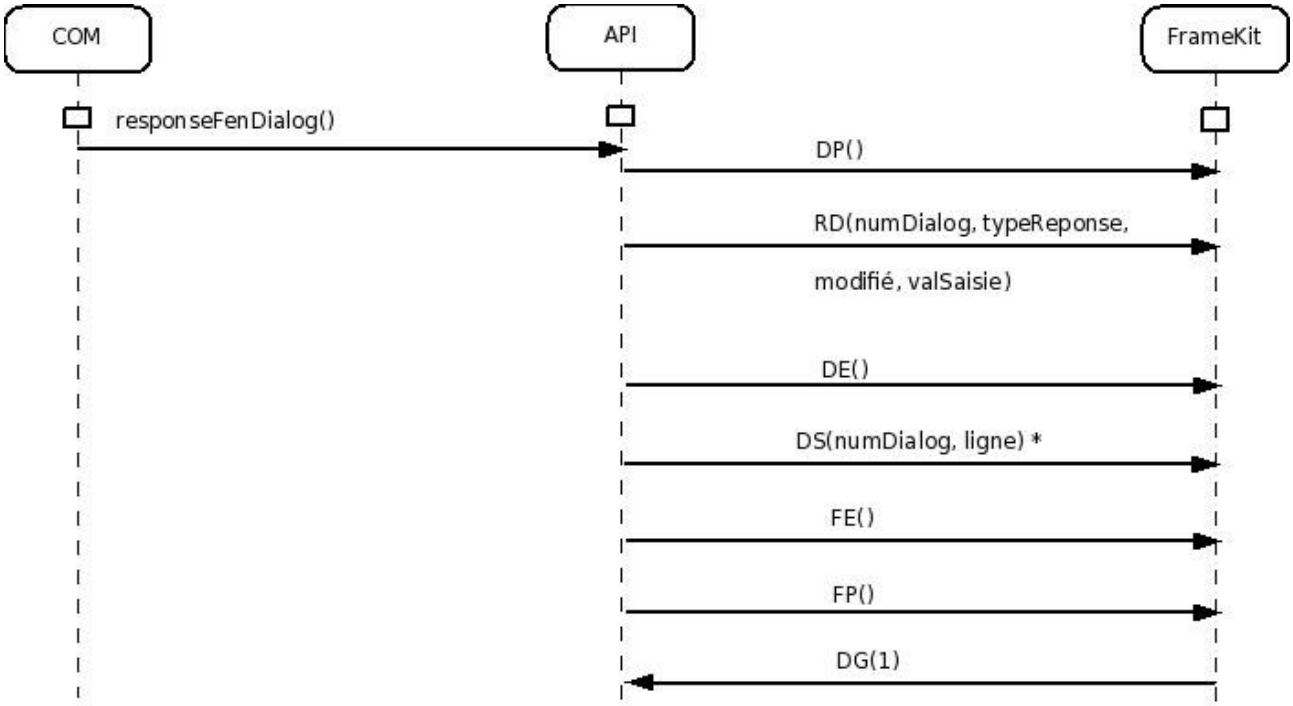

Figure 23: Diagramme de séquence réponse aux dialogues.

Informations échangées

- DP() : début d'une réponse à un dialogue.
- RD : réponse du dialogue numDialog.
  - numDialog : Numéro du dialogue.
  - typeReponse : type de la réponse (OK, Annuler)

- modifié : Modifié ou non.
- valSaisie : valeur de saisie.

- DE : début d'un ensemble de DS.
- DS : suite de dialogue.
  - numDialog : numéro de la boite de dialogue.
  - numLigne : numéro de la ligne dans la boite de dialogue.

- FE : Fin d'un ensemble de DS.
- FP : fin de réponse à un dialogue.
- DG : Destruction du dialogue.

Automate CAMI

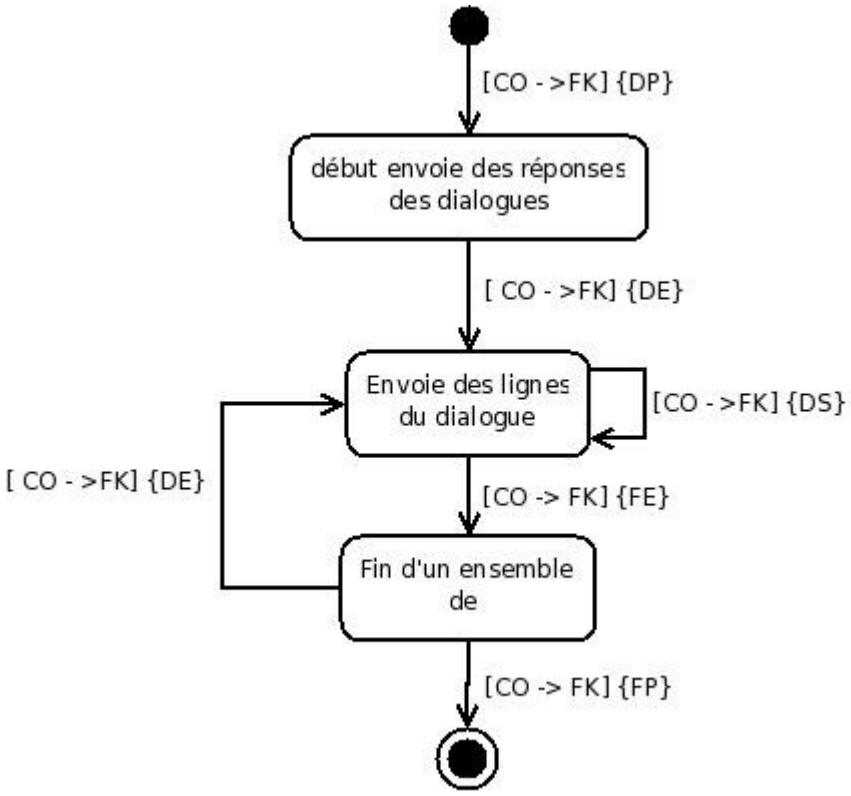

Figure 24 : Automate envoi des réponses.

2. 4. GESTION DES SESSIONS

2. 4. 1. Suspension de la session

Diagramme de séquence

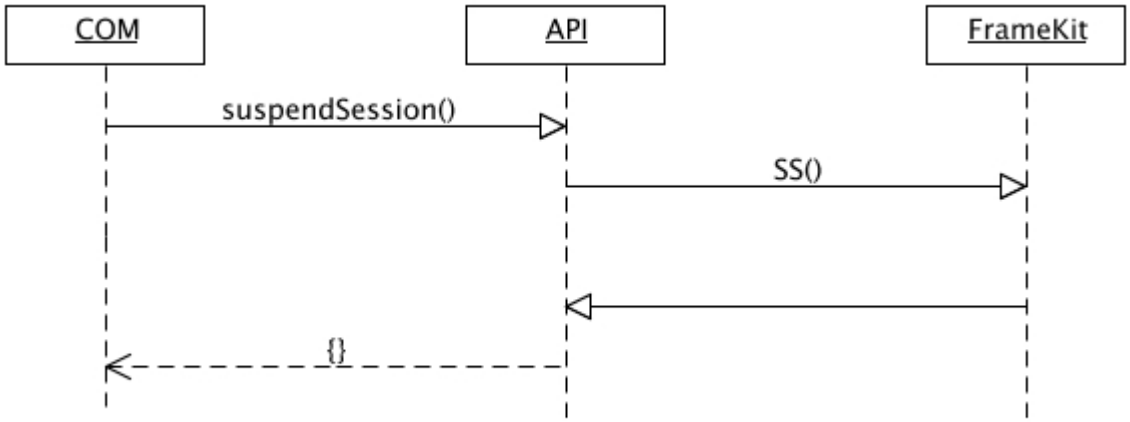

Figure25 : Diagramme de séquence – Suspension de la session.

Informations échangées

- le SS (API --> FK) demande à FK de suspendre la session courante.
- le SS (API <-- FK) accuse la réception d'un SS.

2. 4. 2. Reprise d'une session

Diagramme de séquence

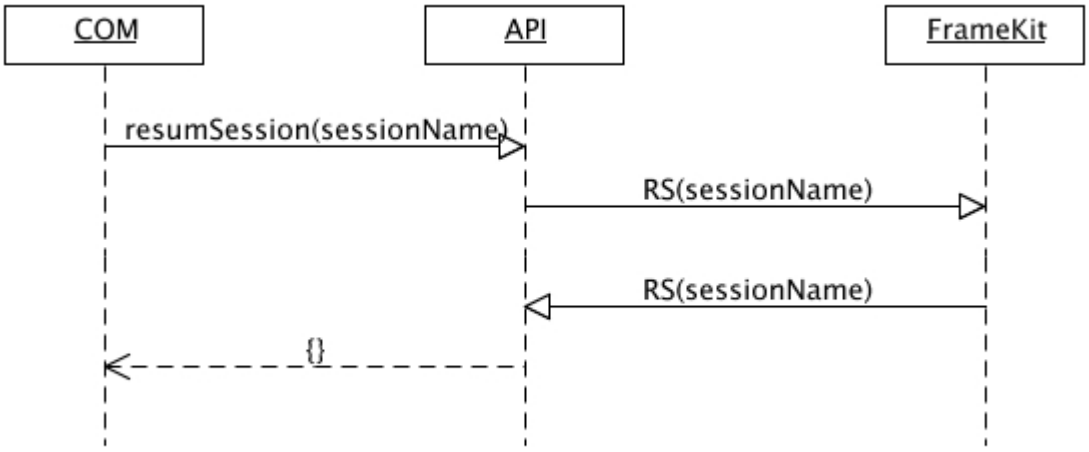

Figure 26: Diagramme de séquence – reprise d'une session.

Informations échangées

- le RS (API --> FK) demande à FK de reprendre la session sessionName.
- le RS (API <-- FK) accuse la réception d'un RS.

2. 4. 3. Fermeture d'une session

Diagramme de séquence

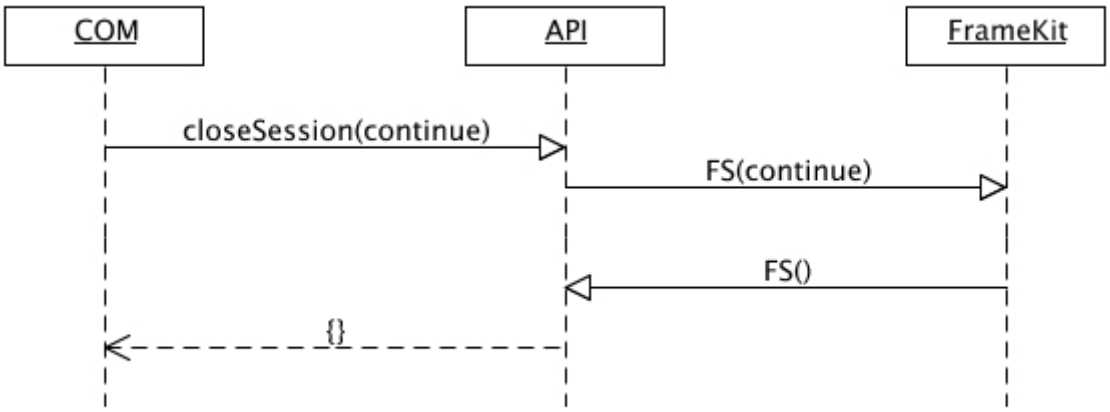

Figure 27 : Diagramme de séquence - Fermeture d'une session.

Informations échangées

- FS (API --> FK) demande à FK de fermer la session courante. « continue » indique si à FrameKit si le calcul doit continuer ou non après la fermeture de la session.
- FS (API <-- FK) accuse la réception d'un FS et ferme la session.

2. 5. Invalidation du modèle et gestion de la date

2. 5. 1. Invalidation du modèle

Diagramme de séquence

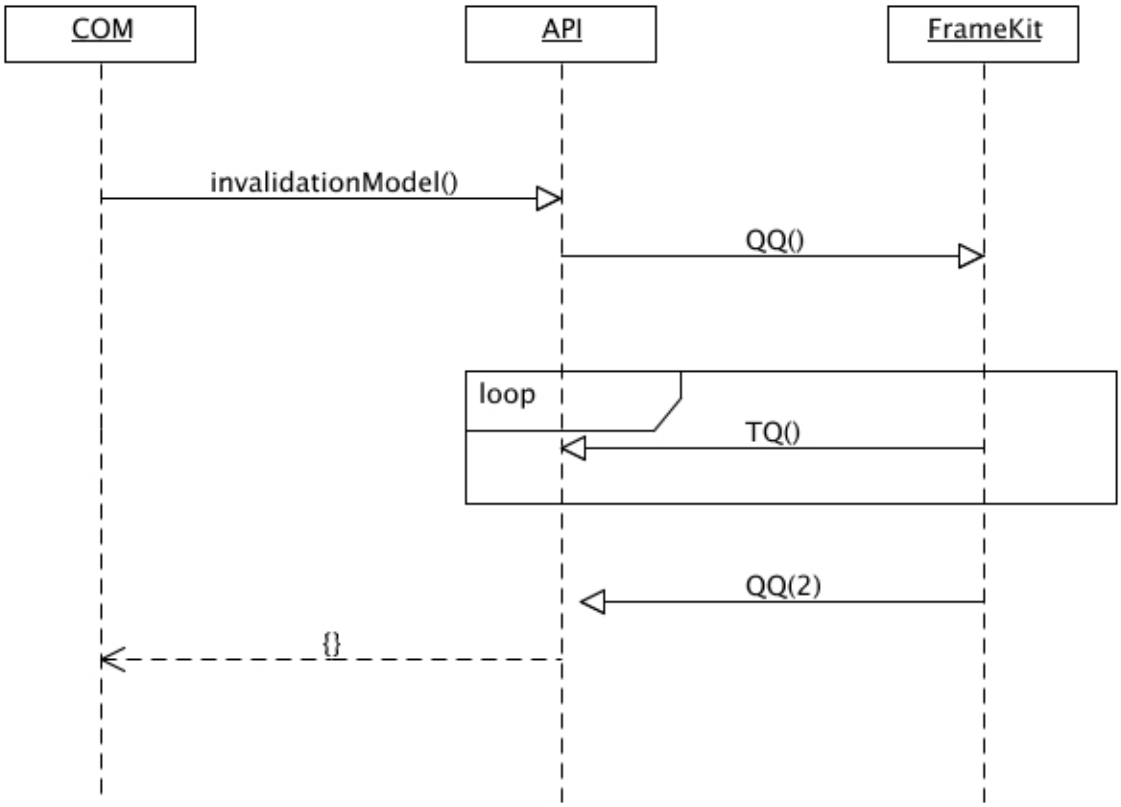

Figure 28: Diagramme de séquence Invalidation du modèle.

Informations échangées

Le QQ() fait une demande d'invalidation du modèle auprès de FrameKit.
Les TQ sont décrits précédemment.
Le QQ(2) indique la fin de l'invalidation du modèle.

Automate CAMI

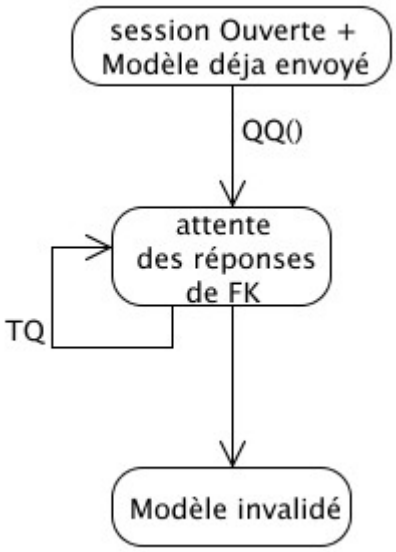

Figure 29: Automate Cami invalidation du modèle.

2. 5. 2. Envoi de la date du modèle à FrameKit

Diagramme de séquence

La date sera envoyée par API à FrameKit lors d'une demande de service suivant une invalidation du modèle.

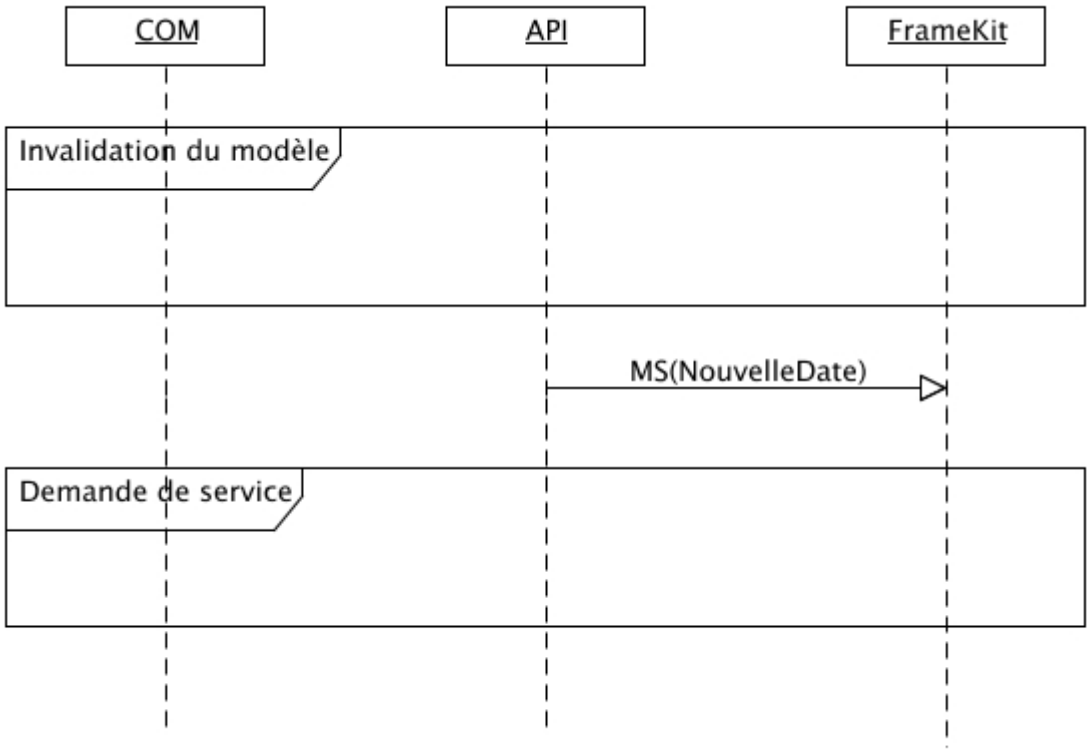

Figure 30 : Envoi de la date du modèle à FrameKit.

Informations échangées
Le MS transmet la nouvelle date sous forme d'un entier à FrameKit.

3. Automate de la session

Le but de l'automate de la session est maintenir l'état des communications dans un état cohérent, suite aux ordres reçus de Coloane et aux aux commandes de FramKit.

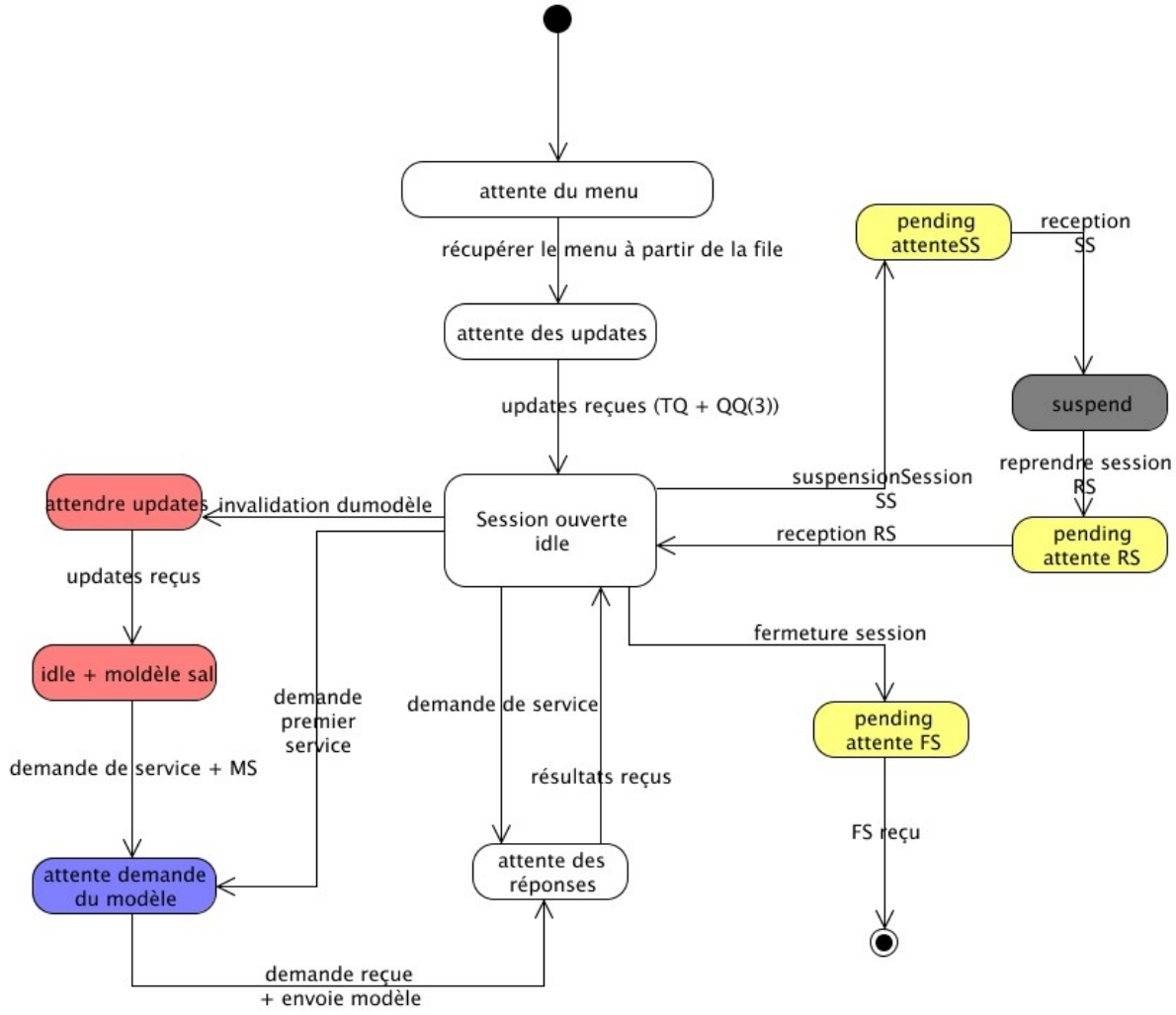

Figure 31: automate d'état de la session

4. Interfaces offertes par API à COM.

4.1. Phase de l'ouverture de session

Pour la représentation des objets entre COM et API dans la phase de l'ouverture de session, on définit les interfaces suivantes.

Pour les objets en entrée

Les informations nécessaires à l'ouverture d'une session (appel de la méthode OpenSession), à savoir le nom de la session, la date et le formalisme de la session. Au niveau de COM, ces informations seront codées dans un objet de classe implémentant l'interface suivante :

```
Interface ISessionInfo{
        String getSessionName(); // nom de la session
        String getSessionDate(); //date du modèle, lors de l'ouverture de la
        session String getSessionFormalism(); //formalisme du modèle
}
```

Objets en sortie

En sortie (retour de la méthode openSession), nous aurons le menu de services et les informations sur FrameKit. Au niveau de API, ces informations seront codées dans un objet de classe implémentant l'interface suivante :

```
Interface IOpenSessionResult{
        IFKInfo getFKInfo(){}
        IMenu getImenu{}
}

Interface IFKInfo{
        String getNameService(); //Nom du service
        String getAboutService(); // Informations à propos du service
        String getIncremental(); // Inrémental
        String getResultatCalcule(); // (1) si fait précédemment (2) non
}

Interface IMenu{
        IMenu getParent(); // Libellé du père de l'élément
        String getName(); //Libellé de l'élément
        int getQuestionType(); //Type de la question
        int getQuestionBehavior(); //Type du choix
        boolean isValid(); //Validation par défaut
        boolean isDialogAllowed(); //Dialogue interdit ou permis
        boolean StopAuthorized(); //Arrêt possible ou pas
        boolean outputFormalism(); // Domaine du résultat
        boolean IsActiv(); //Enable ou disable
        Vector<IMenu> Fils(); //Les sous-menus fils
```

```
}

Interface IAPI{
	IOpenSessionResult OuvrirSession(ISessionInfo info){};
}
```

4.1.format des Objets échangés entre COM et API lors de l'invocation de services

Ensemble de Résultats

```
interface ISetResult{
	vector<subResult> getSubResults();
}

interface ISubResult{
	vector<IRT> getRT();
	vector<IXA> getXA();
	vector<IRO> getRO();
	vector<IME> getME();
	vector<IMT> getMT();
	vector<INode> getNodes();  // CN
	vector<IBoxe> getBoxes();  // CB
	vector<IArc> getArcs(); // CA
	vector<Integer> getObjectsToDelete(); // SU : objets à supprimer
}


interface IRT{
	String getRTText();
}


interface IXA{
	int getObjectID();
	String getAttributeType();
	String getNewContent();
}


interface IRO{
	int getObjectID();
}


interface IME{
	int getObjectID();
}
```

```
interface IMT{
	int getObjectID();
	String getNameAttribute();
	int getIndexBegin();
	int getIndexEnd();
}


interface INode {
	String getNodeType();
	int getNodeID();
	vector<IMonoLineAttribute> getNodeMonoLineAttribute(); // CT
	vector<IMultiLineAttribute> getNodeMultiLineAttribute();  // CM
}

interface IBox {
	String getBoxType();
	int getBoxID();
	int getBoxPage();
	vector<IMonoLineAttribute> getBoxMonoLineAttribute(); // CT
	vector<IMultiLineAttribute>  getBoxMultiLineAttribute(); // CM
}


interface IArc {
	String getArcType();
	int getArcID();
	int getStartNode();
	int getEndNode();
	vector<IMonoLineAttribute> getArcMonoLineAttribute(); // CT
	vector<IMultiLineAttribute>  getArcMultiLineAttribute(); // CM
}


interface IMonoLineAttribute{
	String  getAttributeType();
	String  getContent();
}

interface IMultiLineAttribute{
	String  getAttributeType();
	String  getContent();
	int getLineNumber();
}
```

A partir de là on peut aussi décrire un modèle, qui est un ensemble de noeuds, d'arcs et d'attributs :

```
interface Imodel {
        vector  <INode> getNodes();
        verctor <IBox>   getBoxes();
        verctor <IArc>   getArcs();
}
```

5. Éléments de conception

Architecture

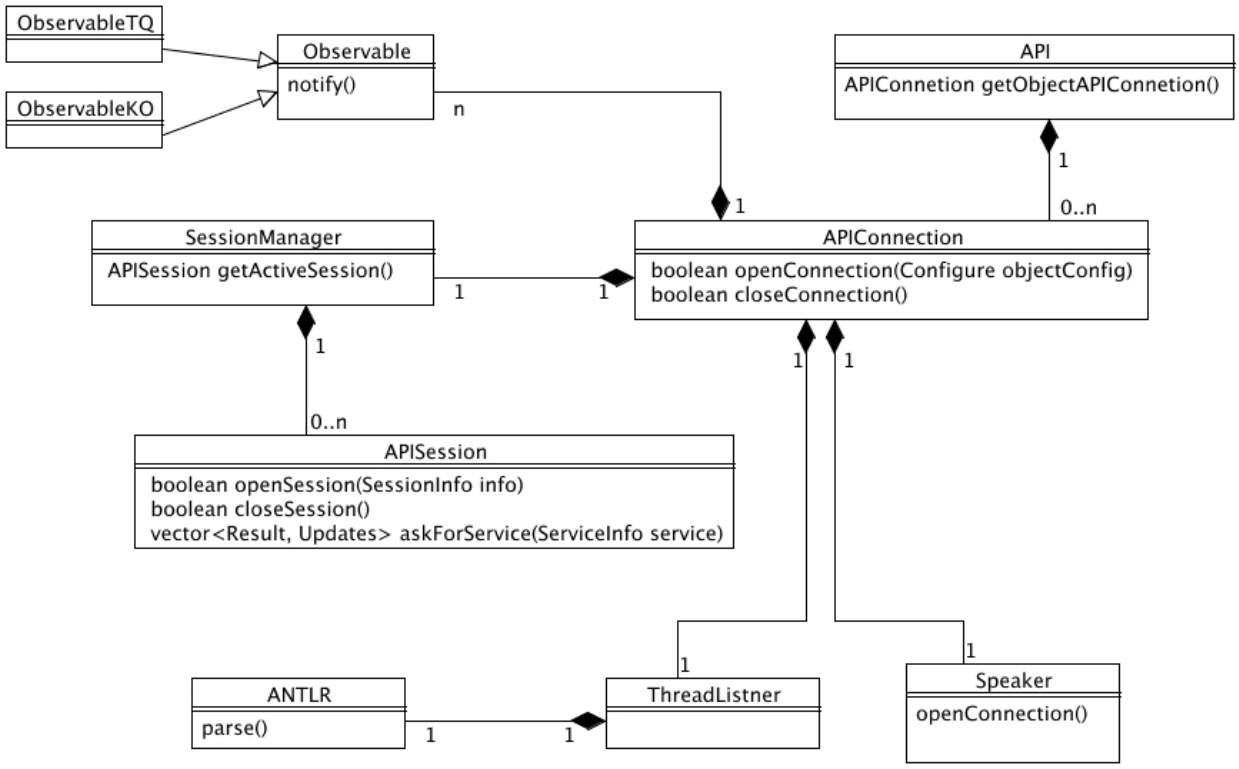

Figure 32: Diagramme de classes de l'API.

La classe API constitue la grosse classe qu'il faut instancier au début pour pouvoir exploiter les services offerts par API.
La méthode getObjectAPIConnection permet d'obtenir un objet de type APIConnection qui servira à n'authentifier et se connecter sur la plateforme.

APIConnection représente la connexion entre Coloane et FrameKit. L'ouverture d'une connexion se fait par appel à la méthode openConnection(). La connexion est rompue par un appel à closeConnection().

APISession est la classe qui représente une session. Pour ouvrir une session, il faut demander à un l'objet APIConnextion un objet de type APISession, et ensuite effectuer une ouverture de connexion sur celui-ci grâce à l'appel openSession().
La classe Speaker se charge de construire les séquences de commandes CAMI

et de les envoyer à FrameKit, il a aussi la charge d'initialiser.

ThreadListner comme son nom l'indique est un thread qui écoute tout ce qui vient de FrameKit.

La classe AntlrParser (dans le fil d'exécution de thread listner) a pour rôle d'analyser les commandes reçus de FrameKit, et de déclancher les traitement à réaliser suite à ces commandes.

La classe SessionManager sera en mesure de gérer les sessions, précisément, le switch (activation/désactivation) entre les sessions.

Les diagrammes de séquence suivants montrent comment interagissent les classes définies ci-dessus.

Demande d'un objet APIConnection

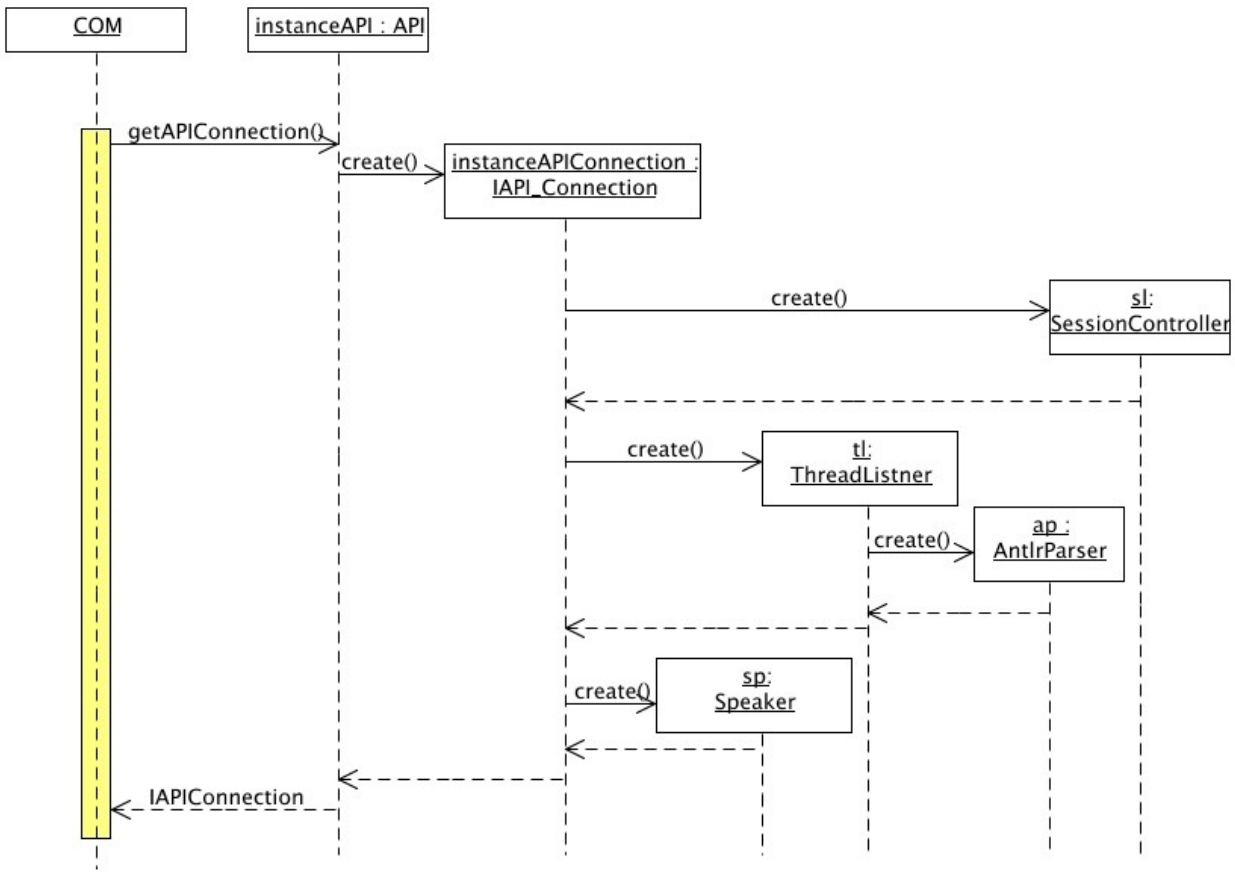

Figure 33 : Demande d'un objet APIConnection.

Ouverture de connexion

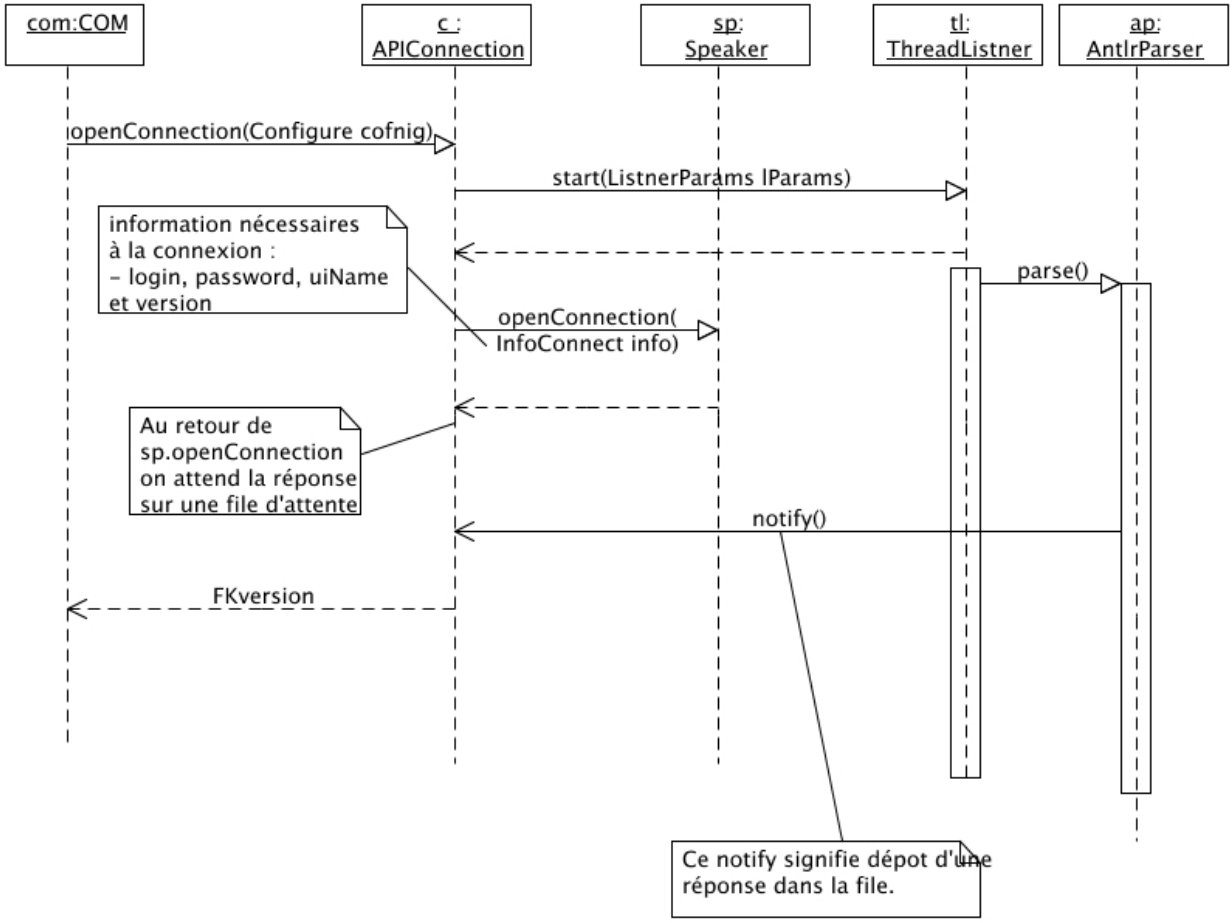

Figure 34: Ouverture d'une connexion.

Ouverture d'une session

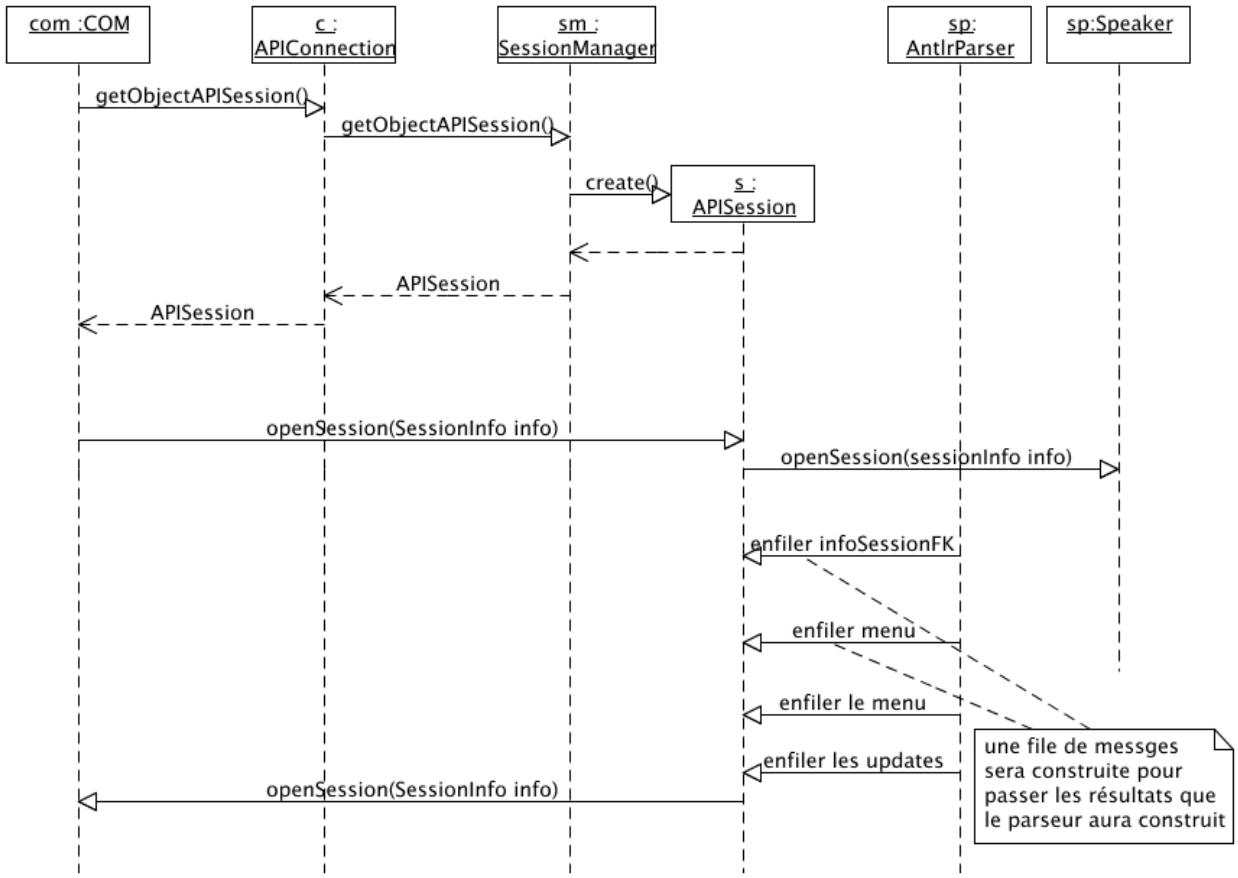

Figure 35: Diagramme de séquence l'ouverture de session.

6. Conception d'un environnement de tests de l'application

Le principe fondamental du test , est de révéler les fautes de l'application afin de permettre l'identification des défaillances.

Tests Unitaires

Ce type de tests sera effectué a la fin de notre application, quand on aura codé toutes nos classes.
Ce procédé permettant de s'assurer du fonctionnement correct d'une partie déterminée d'une portion d'un programme.
A toute interaction doit correspondre un cas de test Junit , cela signifie qu'une nouvelle classe Java doit être construite .
Cette classe doit hériter de la classe Junit TestCase, et doit contenir une méthode correspondant au test .

Tests Fonctionnels

Ce type de tests sera fait pour tester une fonctionnalité parmi celles offertes par notre API, comme demande de service, ou même ouvrir session ...
Dans cette partie là, on aura des diagrammes de séquence UML, qui nous permettra de tester nos fonctionnalités , et pour pouvoir isoler les tests faits sur notre API, sans se préoccuper d'éventuelles erreurs survenant soit de la COM , soit de FrameKit .
Et pour ceci, on construira des Bouchons pour COM et FrameKit, ces bouchons vont simuler leur comportement , mais ce comportement sera codé par nos soins de maniére simple.
Un bouchon va se comporter comme l'objet qu'il simule , or il va se connecter sur le même port , et se mettra a l'écoute
Ces tests seront fait dans les cas ci-dessous:

- *cas nominal : le cas normal, où tout va bien.
- *cas d'erreur : le cas où ça se passe mal, pas de réseau, authentification failure..