Méthodologie de vérification et intégration aux processus de développement

Nous avons présenté, dans le chapitre 2, un état de l'art sur les processus de développement des systèmes embarqués temps-réel. Suite à l'étude de ces processus nous avons fait le constat que peu d'entre eux précisent la manière avec laquelle les activités de vérification formelles sont intégrées aux activités de développement. De ce fait, dans ce chapitre, nous présentons notre méthodologie de vérification permettant l'intégration des activités proposées dans les chapitres précédents aux processus de développement industriel. D'abord, nous présentons les éléments permettant de définir une méthodologie dans la section 8.1, puis nous présentons la méthodologie proprement dite dans la section 8.2.

8.1 Définition et formalisation d'une méthodologie

Le mot méthodologie utilisé dans ce chapitre correspond à la description d'une succession d'activités de travail qui gouvernent la production de modèles formels assimilables par les outils existants de vérification [Colombo *et al.* 2007]. Pour ce qui est du processus de développement, celui-ci correspond aux activités mises en oeuvre au sein d'une équipe de développement pour la production de biens livrables. Ces biens livrables correspondent aux modèles de conception que l'on désire valider formellement afin de s'assurer du respect des exigences exprimées au début du cycle de développement.

8.1.1 Vue d'ensemble de la méthodologie

La méthodologie que nous proposons dans cette thèse intègre les activités liées à la vérification formelle des modèles de conception. En ce sens, elle raffine les méthodologies de conception existantes pour y intégrer les préoccupations de formalisation et de génération de code formel. Le but étant de pouvoir utiliser les outils de vérification formelle existants tout au long du processus de développement. L'approche que nous proposons dans cette thèse a été définie de manière générique pour s'appliquer aux principaux processus de développement utiliser dans le contexte du développement de système logiciels embarqués.

En effet, nous avons présenté dans la section 2.4 deux des processus de développement les plus répandus dans ce contexte : le processus unifié UP [Booch *et al.* 1999] et le processus de développement en "V". À partir de ces deux processus, nous avons dégagé les activités de développement communes afin d'y intégrer les activités proposées dans

notre approche. Ces activités sont : (1) le traitement des exigences (2) Spécification des cas d'utilisation (3) l'analyse et (4) la validation.

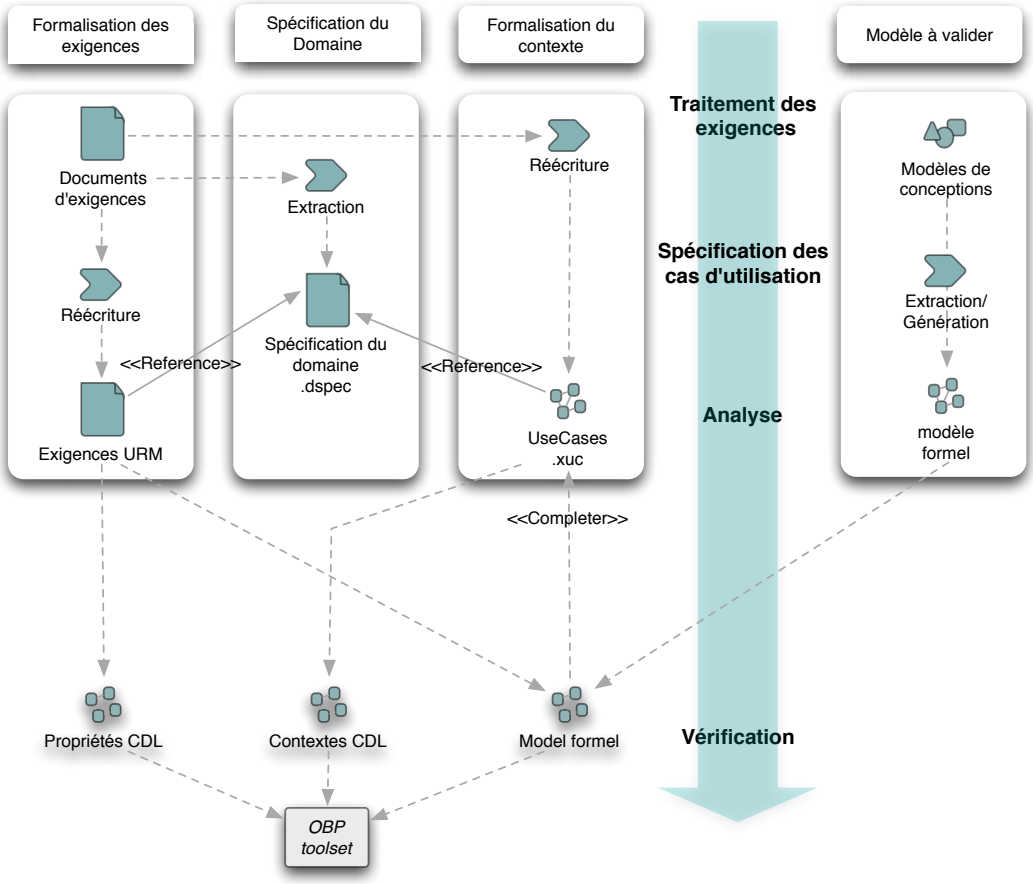

Figure 8.1 : Vue générale de la méthodologie proposée

- **Le traitement des exigences:** cette activité concerne la formalisation des données d'entrées à l'aide des langages proposés dans la deuxième partie de ce mémoire,
- **La spécification des cas d'utilisation:** permet de formaliser les scénarios d'interaction entre le composant du système à vérifier et son contexte,
- **L'analyse:** consiste en la génération de modèles CDL à partir des modèles utilisateurs produits par la phase de traitement des exigences,
- **La vérification:** concerne l'exploitation des outils de vérification formelle pour obtenir le résultat.

Ainsi, la figure 8.1 présente la méthodologie proposée et l'organisation des activités de vérification proposée selon les activités des processus de développement. Le point

d'entrée de celle-ci est un ensemble d'exigences non formalisées ainsi que le modèle de conception préliminaire du système étudié. Ces modèles regroupent les diagrammes de cas d'utilisation et de séquences qu'on trouve dans les documents de spécification pour illustrer et clarifier les exigences sur les fonctionnalités attendues du système. Pour que le modèle à vérifier soit exploité par la méthodologie proposée, celui-ci doit être décrit par un langage formel exploitable par l'outil de vérification formel utilisé. Dans le cas de notre travail, le modèle à validé est présenté sous la forme d'un programme FIACRE pour être vérifier par l'outil TINA. L'aspect formalisation du modèle du système à vérifier n'entre pas dans le périmètre de nos travaux, menés dans le cadre de cette thèse.

Dans la suite de ce chapitre, nous allons détailler chacune des quatre activités méthodologiques. Mais avant de procéder, nous allons présenter la notation de description de méthodologie utilisée dans la section qui suit.

8.1.2 Notation utilisée pour la description de la méthodologie

La notation utilisée dans ce chapitre pour décrire un processus de développement méthodologique étend les diagrammes d'activités UML2 [OMG 2007]. Nous reprenons la syntaxe concrète utilisée dans [Fontan 2008] pour décrire notre méthodologie. En effet, dans ce dernier, les étapes méthodologiques sont décrites par des activités qui peuvent être raffinées en sous activités. Un diagramme d'activité commence obligatoirement par un disque noir représentant le début de la première activité. Chaque étape est reliée par une flèche de transition montrant les relations de précédence entre les activités. Une activité peut être décomposée en sous activités. La figure 8.2 montre un exemple de processus de développement méthodologique.

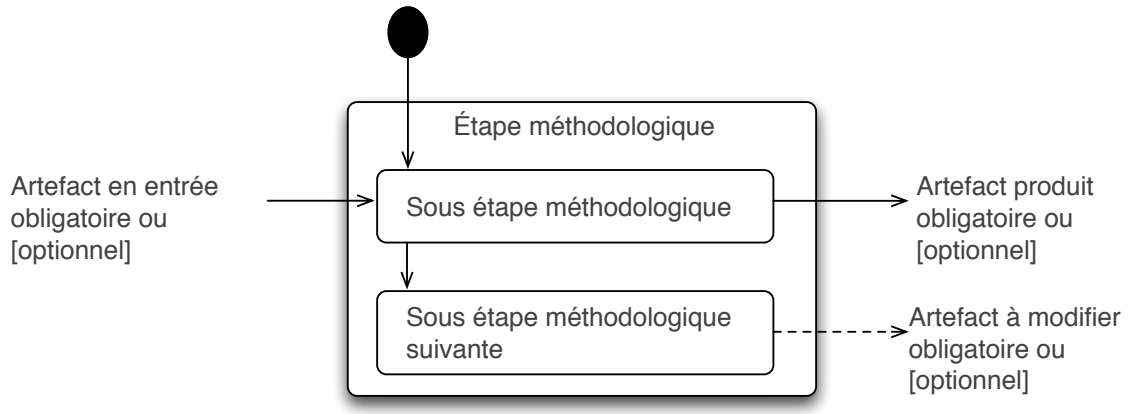

Figure 8.2 : Langage de description de la méthodologie

Lors des étapes de la méthodologie, des artefacts peuvent être produits, consommés ou modifiés par une ou plusieurs étapes. La production (création/modification) et la consommation (consultation) sont respectivement représentées par des flèches pleines (par opposition aux flèches de transition entre activités) sortantes et entrantes de l'activité sur les côtés. Les artefacts modifiés lors d'une activité méthodologique sont représentés par

une flèche sortante marquée en trait discontinu. Un artefact sortant est disponible pour toutes les autres activités, contrairement à la production d'un artefact qui n'est autorisé qu'à l'intérieur des activités qui possèdent une flèche sortante.
Le diagramme d'activités représentant la méthodologie est ainsi composé d'étapes méthodologiques, de référence vers les artefacts utilisés par ces activités et de flèches reliant ces derniers. Ainsi, la méthodologie proposée définit un ensemble d'activités méthodologiques et associe à chacune de ces étapes la livraison d'artefacts. La notion d'artefact ici fait référence à des modèles conformes à des métamodèles.

8.2 Application à la vérification formelle des exigences et l'exploitation des contextes

La méthodologie proposée est dédiée à la vérification formelle de modèles et ne couvre donc pas les phases de conception et d'implémentations. Le but de celle-ci est d'intégrer la production de modèles formels nécessaires à l'utilisation des outils existants de *model checking*. Comme nous l'avons précisé précédemment, nous nous plaçons dans le cadre de la vérification par exploitation de contextes, et notamment en utilisant des modèles CDL ainsi que la chaine d'outils OBP développés dans le laboratoire.

8.2.1 Traitement des exigences

Dans notre approche, la première activité est l'étude des documents d'exigences par l'expert métier afin de les réécrire sous la forme d'exigences URM. Nous supposons que les exigences définies dans ces documents, fournis par le client, sont des exigences de bas niveau (par opposition aux objectifs de haut niveau). En effet, chaque exigence devra porter sur une fonctionnalité simple du système étudié afin de vérifier des contraintes telles que le séquencement d'événements ou des contraintes temporelles sur les messages échangés. Comme nous l'avons préciser au par avant, la décomposition des exigences étant une problématique à part entière, elle n'entre pas dans le cadre de cette thèse.

La figure 8.3 illustre l'activité méthodologique de la phase de traitement des exigences.

8.2.2 Spécification des cas d'utilisation

Cette activité concerne l'exploitation des documents de spécification afin d'identifier les différents cas d'utilisation du système. Ces cas d'utilisation sont ensuite détaillés afin de spécifier les scénarios d'interaction entre le système et les entités de son environnement (acteurs). À ce stade, les scénarios d'interactions peuvent ne référencer que les séquences d'échange nominal entre le système et le contexte. L'information permettant de spécifier de tels scénarios est généralement définie dans les cahiers de charge. Par la suite, ces scénarios peuvent être raffinés afin d'y intégrer les différentes exceptions potentielles qui peuvent altérer le déroulement du scénario nominal. Selon les exceptions identifiées, des *handlers* peuvent être définis pour enrichir la spécification du scénario principal.

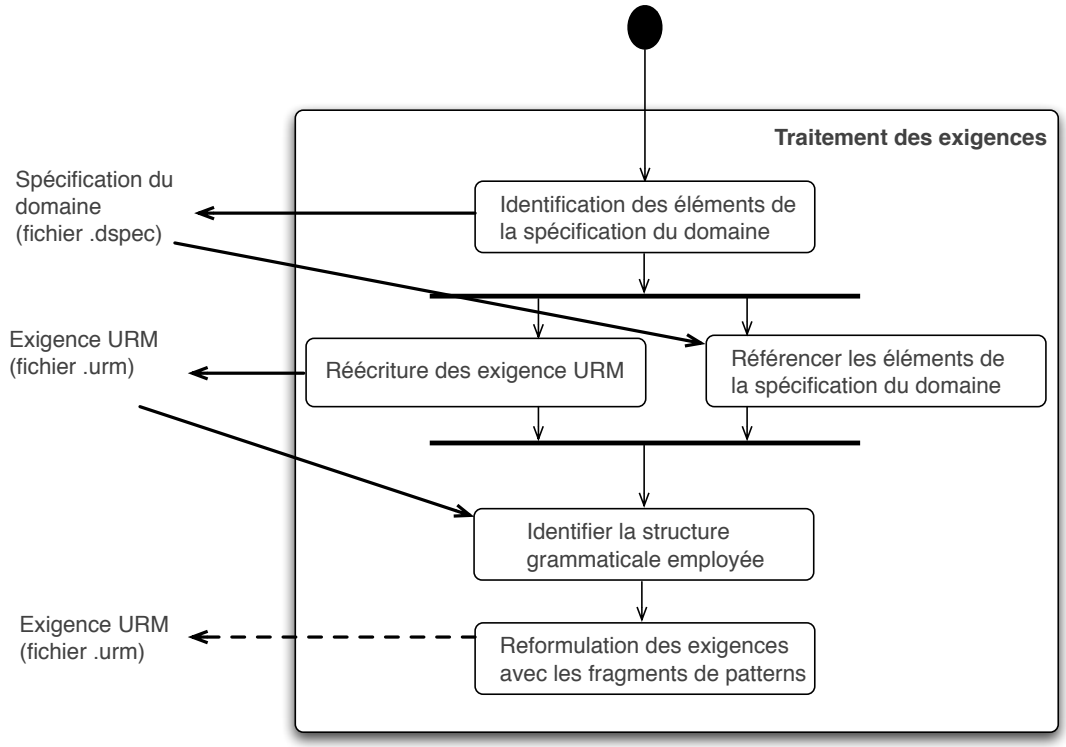

Figure 8.3 : Activité de traitement des exigences

La figure 8.4 détaille cette phase de la méthodologie.

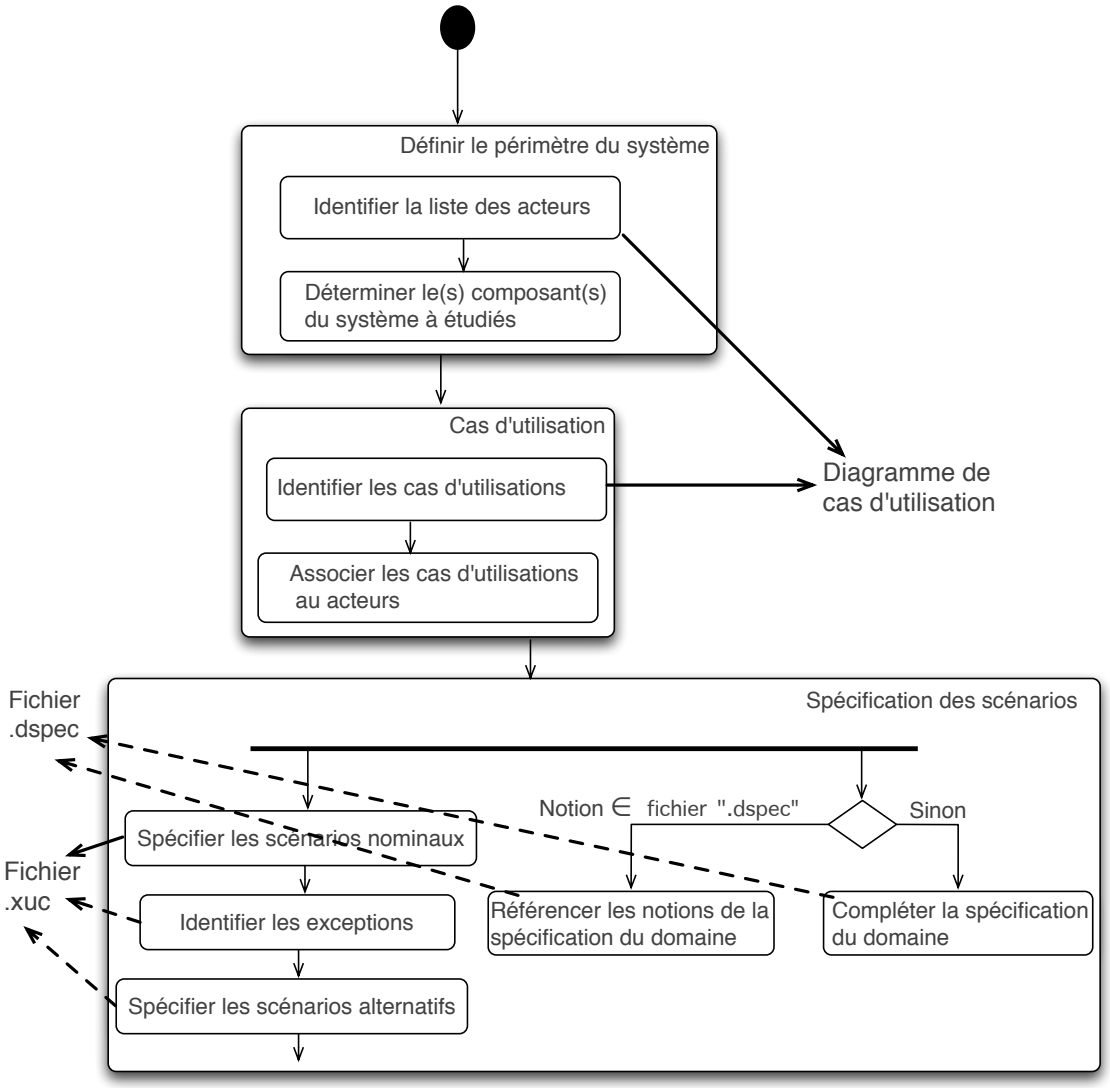

Figure 8.4 : Activité de spécification des cas d'utilisation

8.2.3 Analyse des modèles formels

Cette phase est implémentée sous forme de transformations de modèles permettant la génération de modèles CDL. Les propriétés CDL sont générées à partir des exigences URM issues de la phase de traitement des exigences selon le processus détaillé dans la section 7.2. En ce qui concerne les modèles comportementaux des acteurs, ceux-ci sont obtenus à partir des modèles XUC produits lors de l'activité de spécification des cas d'utilisation. La figure 7.1 présentée au début du chapitre 7 illustre cette activité.

8.2.4 Vérification

La vérification des propriétés formalisées à partir des exigences sur le modèle du système étudié en exploitant les contextes se fait à l'aide de la chaine d'outils OBP [Dhaussy *et al.* 2009]. Au cours de cette phase, les modèles CDL sont traduits, dans l'outil OBP, en codes assimilables par un vérificateur. Dans nos expérimentations, OBP génère des programmes Fiacre [Berthomieu *et al.* 2008] qui sont exploités soit par l'explorateur OBP (*OBP Explorer*) soit par l'outil TINA [Berthomieu *et al.* 2004]. Ensuit, OBP récupère les résultats retournés par ces composants afin d'évaluer la correction de la propriété. La figure 8.5 présente une vue d'ensemble de cette chaine d'outils.

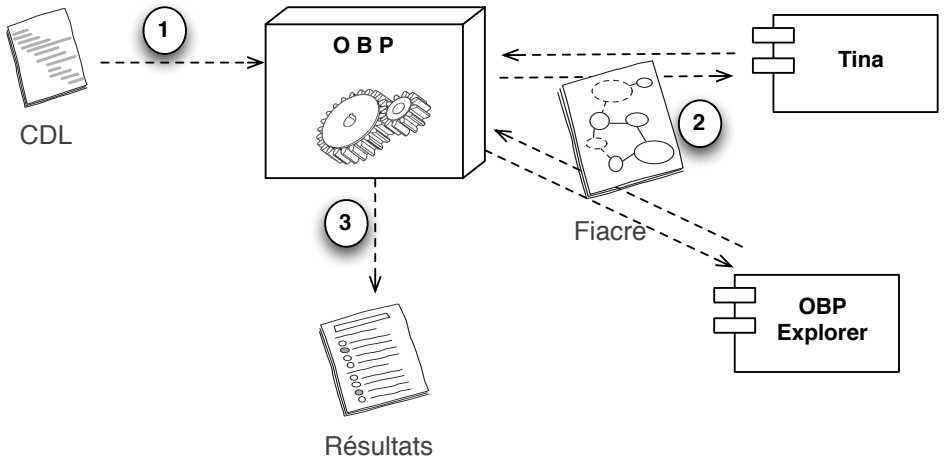

Figure 8.5 : L'outil OBP

8.3 Discussion et synthèse

Nous avons présenté dans ce chapitre une méthodologie permettant l'intégration des activités liées à la vérification formelle par exploitation de contextes dans un processus de développement logiciel. L'originalité de notre méthodologie consiste à proposer aux experts métier des formalismes adaptés à la formalisation des contextes et des exigences dès les premières étapes du processus de développement. Ces formalismes, que nous avons présentés dans la deuxième partie de ce mémoire, réduisent le gap entre les modèles manipulés lors des premières étapes de développement et ceux nécessaires à la vérification formelle. Ainsi, au cours de cette méthodologie, les modèles de contextes et des exigences peuvent être raffinés de façon incrémentale à partir des cas d'utilisation et des exigences textuelles.

Aussi, la chaîne d'outils OBP constitue une interface commune pour la génération de code formel vers le *model checker* ciblé. En effet, OBP intègre les transformations de modèles nécessaires pour générer, à partir d'un même modèle CDL soit du code FIACRE pour utiliser l'outil TINA, soit du code IF [Bozga *et al.* 1999] afin d'exploiter le simulateur IFx. Les retours de preuve d'OBP ne sont pas pris en charge dans notre approche. Toutefois,

ce travail est en cours au sein d'équipe IDM de l'ENSTA-Bretagne afin d'intégrer la prise en charge de ces retours et de les présenter aux utilisateurs directement dans les modèles de haut niveau.

Nous évaluons la pertinence et l'applicabilité de notre méthodologie pour valider le modèle d'un composant logiciel industriel réel. Le chapitre suivant présente cette évaluation.

EXPÉRIMENTATIONS ET CONCLUSION

TELECOM Bretagne

9 Cas d'Étude Industriel: Validation du composant AFS_SM

Nous appliquons l'approche de vérification proposée dans ce mémoire sur le cas d'étude présenté dans le chapitre 3. Ce cas d'étude a été utilisé le long de ce mémoire afin d'illustrer les concepts et les algorithmes proposés. Dans ce chapitre, nous allons détailler l'ensemble des étapes de la méthodologie proposée dans le chapitre précédent pour vérifier les exigences du système en considérant ses cas d'utilisation. Le but de cette expérimentation est de montrer l'applicabilité de notre approche et la contribution apportée pour améliorer l'intégration des activités de vérification formelle aux processus de développement utilisés chez notre partenaire industriel. Ce chapitre est constitué de trois parties : La première partie est une description des exigences et des cas d'utilisation fournis en entrée par notre partenaire industriel. La deuxième partie est une application de la méthodologie de vérification présentée dans ce manuscrit sur le cas d'étude. Dans la troisième partie, nous étudions l'apport de notre approche par rapport à une application classique des techniques de vérification formelle.

9.1 Formalisation des cas d'utilisation

Cette section commence par la présentation les artefacts fournis par notre partenaire industriel pour la validation du composant SM du système AFS. Puis, nous présentons les différents cas d'utilisation et le processus de leur formalisation à l'aide du langage XUC. Ce processus de formalisation débouche sur la génération de la partie contexte des programmes CDL. Celle-ci est utilisée, par la suite, pour la vérification des propriétés issues des exigences. Cette dernière étape est détailler dans la section 9.2.

Une description des différents composants du système a été faite dans la section 3.1. Ce chapitre, se focalise sur les cas d'utilisations formalisés à travers le langage XUC ainsi que les exigences traitées.

En effet, les fonctions techniques du SM ont pour objectif de fournir les capacités opérationnelles pour répondre aux exigences des utilisateurs. Parmi ces fonctions techniques ont trouve:

- Lancer une mission
- Arrêter une mission
- Arrêter le système

- Gérer les opérations conduites par les opérateurs: *login*, *logout*, sélectionner une mission, sélectionner un rôle....

Le diagramme des cas d'utilisation du système, présenté dans la spécification technique est présentée dans la figure 9.1.

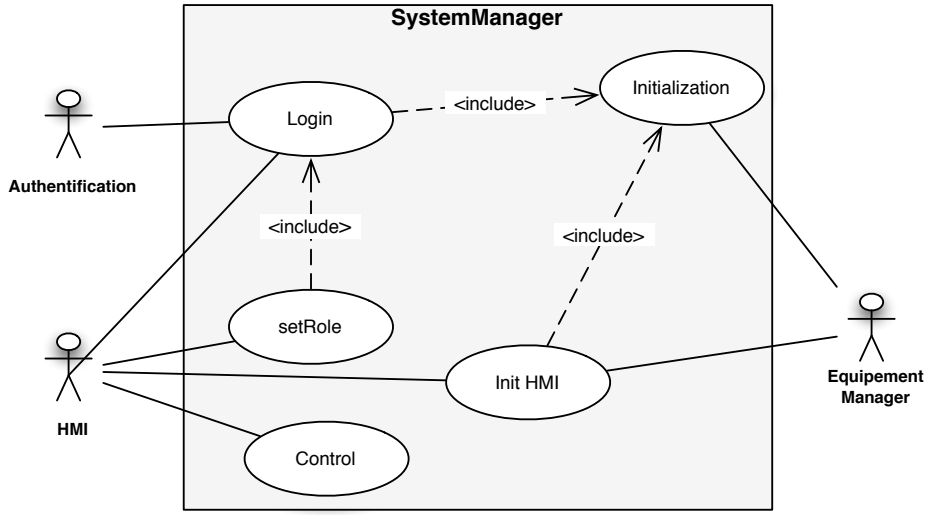

Figure 9.1 : Cas d'utilisation du composant *SM*

Le système AFS a été modélisé par les ingénieurs de notre partenaire industriel en utilisant l'outil *Rhapsody*. Par la suite, une implémentation Java a été générée depuis les modèles *Rhapsody*. La description des scénarios d'interactions entre le système et son contexte pour la réalisation des différentes fonctions techniques est présentée dans la spécification technique sous la forme de diagramme de séquence *Rhapsody*. Un exemple de ces diagrammes est présenté à la figure 9.2.

Dans ce diagramme, on trouve la description des scénarios d'interaction liés à trois cas d'utilisation:

- **Initialisation:** l'interaction correspondante à ce scénario est référencée en utilisant le mot clé *"Ref"* proposé par le standard UML2,
- **Login:** permettant l'authentification des utilisateurs par identifiant et mot de passe. Ce scénario débouche sur le changement d'état du composant *SM* qui passe de l'état *Idle* à l'état *Logged*,
- **Sélection de missions:** (*setRole*) permettant d'attribuer une mission (donc un rôle) à l'utilisateur authentifié.

Donc, un premier travail consiste à identifier les interactions propres à chaque cas d'utilisation. Ainsi, la formalisation de ce scénario d'interaction est réalisée en formalisant les trois cas d'utilisations référencés en utilisant le langage XUC.

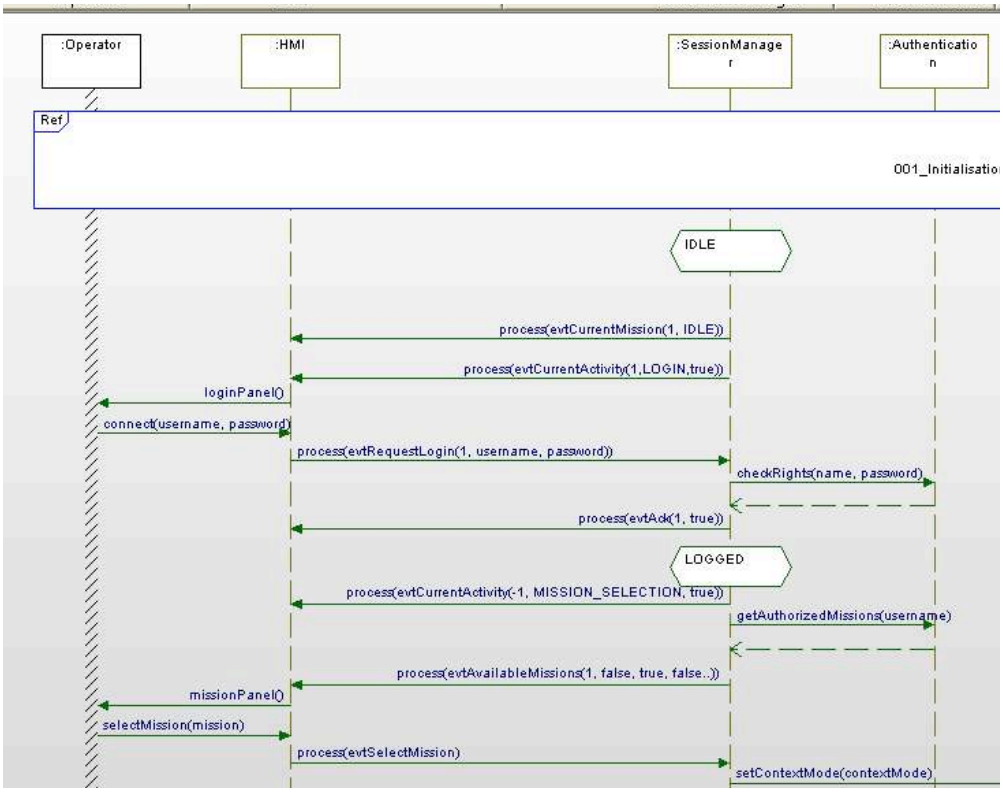

Figure 9.2 : Exemple de scénario fournis dans les documents de spécification

En effet, le composant *SM* permet aux opérateurs de lancer une mission, d'arrêter une mission ou d'éteindre le système. Il est aussi responsable de la gestion des profils des utilisateurs (contrôle d'accès, attribution des rôles). Les opérateurs peuvent interagir avec ce composant en envoyant des requêtes à travers des consoles appelées *HMI*. Les consoles envoient des requêtes au composant *SM* au moyen de communications asynchrones. Ce dernier répond par l'affirmative ou par la négative et renseigne au besoin les *HMIs* de l'évolution du système. Dans certains cas d'utilisation, le *SM* doit communiquer avec d'autres acteurs, comme l'authentification (Authentification) ou le gestionnaire d'équipements (*Equipement Manager*).

À titre d'exemple, les requêtes envoyées au SM via les HMIs sont :

- Se connecter
- Se déconnecter
- Sélectionner une mission (avec un contexte guerre/paix et un mode combat/entrainement)
- Sélectionner un ou plusieurs rôles
- Stopper une mission en cours
- Passer de la préparation à l'opération et inversement

- Eteindre le système

Quant aux informations envoyées par le *SM*, on trouve:

- Confirmation (« acknowledgement »)
- Activité courante
- Mission courante
- Les requêtes accessibles

L'ensemble de ces interactions ainsi que les arguments échangés ont été répertoriés au sein d'une spécification du domaine propre au composant SM du système AFS. Cette spécification du domaine servira aussi lors de la phase de réécriture des exigences en langage URM. Au cours de nos expérimentations, cette spécification du domaine a été construite manuellement à partir de la documentation technique du système ainsi qu'à partir des modèles UML (diagrammes de classes et des machines à états). Une vue partielle de cette spécification du domaine construite pour ce cas d'étude est présentée à la figure 9.3.

SystemManager.dspec

```
DomaineSpecification
{
    Actors{
        EquipmentMgr:: 'Gestionnaire d"equippepments'
        Authentification:: 'Gestionnaire d"authentification'
        HMI:: 'Console'
    }

    SystemElements{
        SM:'System Manager'
    }

    Notions{
        //Consoles <-> SM
        Message login :'demande de login des console vers le SM'
        Message ackLoginOK :'acquittement du login reussit'
        Message ackLoginKO :'acquittement du login échoué'
        Message currentUser : 'spécifier l"utilisateur courant'
        Message logout :'demande de logout des console vers le SM'
        Message ackLogout :'acquittement du logout'
        Message EvtRequestMissionPhaseHmi : 'Demande de mission'
        Message EvtAckHmi : 'Acquitement des HMI'
        Message EvtMissionPhaseHmi : 'Reponse de mission'
        Message currentUserID : 'Spécifier l"identifiant de l"utilisateur courrant'
        Message currentUserMission : 'Spécifier l"identifiant de la mission courran
        //Authentification <-> SM
        Message check : 'vérifier les identifiants'
        Message ackCheckOK : 'acquittement reussit de la vérification des identific
        Message ackCheckKO : 'acquittement échoué de la vérification des identifica

        State Active_Preparation : 'Active preparation state'
        State Active_Operational : 'Active operational state'
        State Operational : 'Operational state'
```

Figure 9.3 : Spécification du domaine spécifique au composant SM

La construction des modèles de contexte a nécessité l'étudie les artefacts décrivant les cas d'utilisation du composant SM. Nous disposions au départ de l'étude :

- De diagrammes de Use Cases UML Rhapsody
- De diagrammes de séquence UML Rhapsody

- De machines à états UML Rhapsody
- De séquences JUnit
- Du code Java
- D'exigences textuelles.

Toutefois, nous avons aussi constaté que les diagrammes UML n'étaient pas à jour par rapport au code java fourni. En effet, les codes java générés ont été manuellement modifiés. Nous nous sommes donc basés sur ce code java et notamment sur les séquences JUnit afin d'identifier les interactions entre le composant SM et son contexte.

Ainsi, sur les cinq cas d'utilisation présentés dans la figure 9.1, nous avons spécifié les scénarios nominaux ainsi que les différentes exceptions que peut rencontrer le composant SM au cours de son cycle de vie. La figure 9.4 présente les trois XUC correspondant au scénario de la figure 9.2. Nous avons utilisé la relation *"include"* entre les XUC (présentée à la section 5.3).

```
XUC login

body {
    include Initialization
    step s1: HMI sends evtRequestLogin to SM
    step s2: SM sends checkRights to Authentification
    step s3: SM receives ackEvtRequestLogin from Authentification
    step s4: SM sends evtAvaliableMissions to HMI
}

exception badIdentifier{
    description 'The user enter wrong identifier/password three times'
    relatedStep s2
    relatedHandler badIdentifierHandler
    next s1
    outcome failure
}

handler badIdentifierHandler{
    step h1: SM sends ErrorFailedMessage to HMI
    step h2: SM sends LockAccount to Authentification
}
```

```
XUC setRole

body {
    include login
    step s1: SM sends evtCurrentActivity to HMI
    step s2: SM receives ackevtCurrentActivity from HMI
    step s3: SM sends evtAvalibleMissions to HMI
    step s4: HMI sends evtSelectMission to SM
    step s5: SM sends setContextMode to Authentification
}
```

```
XUC Initialization

body{
    step s1: HMI sends login to SM
    step s2: SM sends check to Authentification
    step s3: Authentification sends ackCheckOK to SM
    step s4: SM sends ackLoginOK to HMI
    step s5: SM sends currentUserID to HMI
    step s6: SM sends currentUserMission to HMI
}

exception MessageLost{
    description 'In that case that the response to a message
                 is not received within d_max duration'
    relatedStep s3,s4
    relatedHandler MessageLostHandler
    next s2
}

handler MessageLostHandler{
    step h1: SM sends check to Authentification
}
```

Figure 9.4 : Exemple de modèles XUC construit pour le SM

Suite à cette spécification, nous avons appliqué les algorithmes présentés dans les chapitres précédents afin de générer les modèles CDL décrivant les comportements des acteurs *HMI*, *Authentification* et *Equipment Manager*.

L'ensemble des cas d'utilisation de la figure 9.1 ont été formalisés en utilisant le langage XUC pour décrire les interactions entre le SM et son environnement. Au cours de cette formalisation, les différentes exceptions détaillées dans la documentation de

spécification ont été intégrées aux scénarios concernés. De ce fait, pour les cinq cas d'utilisation présentés dans la figure, 12 modèles CDL ont été générés. Ces modèles correspondent aux comportements des trois acteurs participants dans les différents cas d'utilisation.

En effet, pour la génération de la partie contexte, les modèles XUC ont été traduits en modèles UML à travers une transformation en deux temps; (1) traduction des scénarios XUC en diagrammes d'activités (un seul diagramme d'activité généré par XUC), puis (2) séparation des comportements des acteurs participant au cas d'utilisation dans des diagrammes d'activités séparés. Ces deux transformations ont été implémentées en langage de transformation QVT [OMG 2008a] comme nous l'avons illustrer dans la figure 7.1.

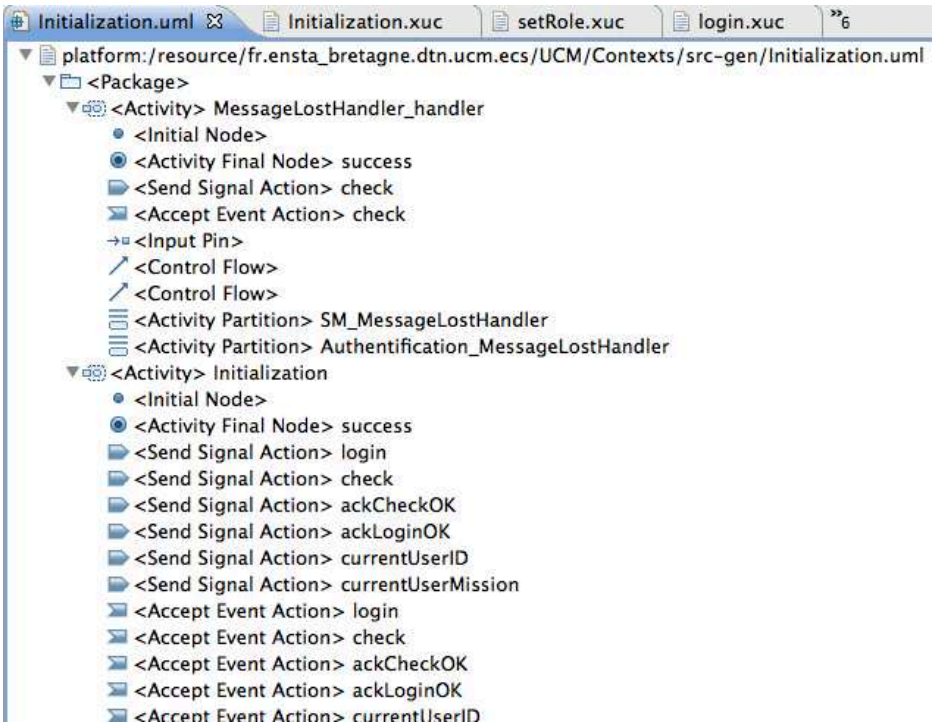

Figure 9.5 : Diagramme d'activité du XUC *Initialization*

La figure 9.5, présente les diagrammes d'activité générés pour le XUC *Initialization* de la figure 9.4. En effet, on avons généré un diagramme d'activité par scénario: un pour le scénario principale et un autre pour le scénarios décrit par le *handler* (comme nous l'avons expliquer à la section 7.1.2). Ces deux diagrammes représentent le comportement globale du XUC correspondant.

La deuxième transformation nous génère un diagramme d'activité pour chaque acteur participant au cas d'utilisation selon l'algorithme présenté à la section 7.1.2.2. La figure 9.6 présente le modèle UML décrivant les activités générés.

La dernière étape consiste à traduire les diagrammes d'activités représentant les comportements des acteurs en programmes CDL. Cette transformation, modèle vers texte, a

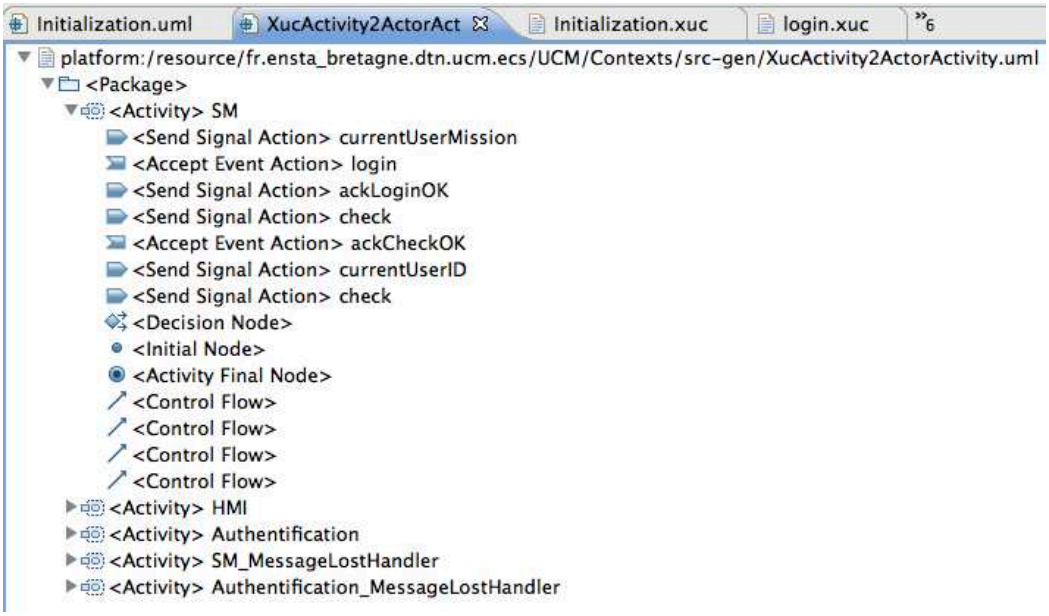

Figure 9.6 : Modèle UML généré par la deuxième transformation

été implémenté à l'aide du langage de génération de code *Xpand*[1].

9.2 FORMALISATION DES EXIGENCES

Les exigences fournies dans les cahiers des charges sont au nombre de 70. La première colonne du tableau 9.1 présente les dix exigences traitées lors de notre expérimentation. En effet, plusieurs exigences rencontrées dans le cahier des charges se ressemblent d'un point de vue structurel, et donc leur formalisation suit le même schéma. Nous avons choisi ce sous-ensemble d'exigence de façon à ce qu'il soit représentatif de l'ensemble des exigences exprimées.

Dans cette phase, nous avons réécrit les exigences présentées dans le tableau 9.1 afin de préparer la génération automatique des propriétés CDL. Puis, nous avons appliqué le processus de spécification des exigences présenté dans la figure 7.4 à la page 103.

Cette activité démarre par l'identification des éléments de la spécification du domaine. Celle-ci fournira en sortie un ou plusieurs fichiers DSpec listant les différentes notions du domaine. Lors de cette phase, nous avons utilisé l'éditeur des exigences URM, développé dans le cadre de cette thèse, afin de réécrire les exigences présentées dans le cahier des charges. Comme le montre la figure 8.3, une spécification du domaine est utilisée au cours de cette phase. Cette étape aboutit à la construction d'une base de données répertoriant les entités et notions du domaine étudié. Cette base de données est présentée sous la forme d'un ou plusieurs fichiers *dspec* qui regroupent l'ensemble des entités du domaine que l'on peut trouver dans les exigences. Cette étape est effectuée en parallèle avec la réécriture des exigences en langage URM. De cette façon, l'utilisateur construit la spécification du

[1] http://www.eclipse.org/modeling/m2t/?project=xpand

domaine au fur et à mesure de la réécriture des exigences. À chaque fois qu'un message ou un argument de message est utilisé dans l'exigence à réécrire, ce dernier l'ajoute dans le fichier DSpec puis le référence dans son exigence URM.

Il est à noter que selon la méthodologie proposée dans ce mémoire, les activités de formalisation des exigences et celle de la formalisation des cas d'utilisation peuvent démarrer indépendamment ou en parallèle, sans un ordre spécifique. De ce fait, on peut réutiliser et potentiellement étendre la spécification du domaine issue de l'activité de spécification des cas d'utilisation si celle-ci a déjà entamé la définition d'une spécification du domaine.

Par la suite, nous utilisons les fragments de patrons de propriétés pour remplacer les termes utiliser dans les exigences par les mots clés listés dans le tableau 7.1. Les exigences issues de ces deux étapes manuelles de réécriture et de consolidation sont regroupées dans le tableau 9.1.

Ainsi, lors de cette phase de réécriture des exigences, pour chacune des exigences, nous avons procédé comme suit:

- Identification des éléments du domaine: Nous avons identifié les éléments appartenant au domaine du composant à valider et nous les avons ajouté à la spécification du domaine produite sous la forme du fichier *SystemManager.dspec*. Ces termes apparaissent dans le tableau 9.1 en soulignés.

- Dans la colonne "Exigences URM", nous avons réécrit les exigences des cahiers des charges en utilisant les mots clés d'affectation et les expressions logiques (AND, OR, WHEN, WITH, =...) présentées dans la section 7.2.1.

- Au cours de la consolidation des exigences, nous utilisons les fragments de patrons proposés dans le tableau 7.1 pour réécrire l'exigence URM. Celle-ci est ensuite analysable par notre algorithme qui détermine quel patron de propriété CDL faut-il instancier pour la génération de la propriété CDL.

La génération des propriétés CDL est obtenue par identification du patron de propriétés à appliquer pour chaque exigence URM. En effet, les exigences URM issues de la phase de consolidation des exigences (section 7.2.2) sont analysées l'algorithme d'analyse des exigences (présenté à la section 7.2.3) afin d'identifier la structure utiliser pour la formalisation de l'exigence. Ainsi, pour chaque exigence, nous avons identifié le patron de spécification de propriété nécessaire à la génération de la propriété CDL. Nous avons utilisé le langage de génération de code *Xpand* avec les templates de propriétés CDL, présentés à la figure 7.8, pour la génération des propriétés CDL.

Table 9.1 : Exigences sur le composant SM du cas d'étude AFS 1/3

ID	Exigence de départ	Exigences URM	Exigence consolidées
473	If the S_CP is in Active_Preparation state, upon reception of a valid EvtRequestMissionPhaseHmi from the HMI with operationPhase to OPERATION, the S_CP shall: send an EvtAckHmi with result to TRUE to the HMI.	If **STATE**(S_CP) = Active_Preparation, **AND** S_CP **recieves** EvtRequestMissionPhaseHmi **FROM** HMI **WITH** operationPhase=OPERATION, **THEN** S_CP **send** EvtAckHmi **WITH** result=TRUE to HMI.	It is always the case that If **P** holds THEN **S** eventually holds. **With:** **P** =[STATE(S_CP) = Active_Preparation & S_CP recieves EvtRequestMissionPhaseHmi FROM HMI WITH operationPhase=OPERATION] **S** =[S_CP send EvtAckHmi WITH result=TRUE to HMI]
461	If the S_CP is in Active_Preparation state, upon reception of an EvtRequestMissionPhaseHmi with operationPhase to OPERATION, the S_CP shall refuse the state transition if the S_CP is restoring an AFSOperationalConfiguration.	If **STATE**(S_CP) = Active_Preparation, **AND** S_CP **recieves** EvtRequestMissionPhaseHmi **FROM** HMI **WITH** operationPhase=OPERATION, **IF** AFSOperationalConfiguration = restoring **THEN** S_CP **shall** refuse the state transition.	It is never the case that **P** holds **IF S** eventually holds. **With:** **P** =[STATE(S_CP) != Active_Preparation **S** =[AFSOperationalConfiguration=restoring] & S_CP recieves EvtRequestMissionPhaseHmi FROM HMI WITH operationPhase=OPERATION]
008	If S_CP is in state Init, upon state change of S_CP to Standby, the S_CP shall send EvtCurrentMissionHmi(IDLE) to each HMI in order to set the mission to idle and Send EvtCurrentActivityHmi(LOGIN, true) to each HMI in order to activate login	If **STATE**(S_CP) = Init, **AND** S_CP **recieves** StateChangeRequest **FROM** HMI **WITH** State=Init, **THEN** S_CP **send** EvtCurrentMissionHmi **WITH** LOGIN=TRUE to **ALL** HMI.	It is always the case that If **P** holds THEN **S** eventually holds. **With:** **P** =[STATE(S_CP) = Init & S_CP recieves StateChangeRequest FROM HMI WITH State=Init] **S** =[S_CP send EvtCurrentMissionHmi WITH LOGIN=TRUE to **ALL** HMI]

[a] Les termes soulignés représentent des références vers des éléments de la spécification du domaine

Table 9.1 : Exigences sur le composant SM du cas d'étude AFS 2/3 (continued)

ID	Exigence de départ	Exigences URM	Exigence consolidées
020	If the S_CP is in Standby state and if the mission is not yet selected, upon reception of a EvtRequestLoginHmi message from the HMI, the S_CP shall send to the HMI an EvtAckHmi message with result to TRUE	If **STATE**(S_CP) = Standby, **AND** isMissionSelected = FALSE **AND** S_CP **recieves** EvtRequestLoginHmi **FROM** HMI, **THEN** S_CP **send** EvtAckHmi **WITH** result=TRUE to **ALL** HMI.	It is always the case that If **P** holds THEN **S** eventually holds. **With: P** =[STATE(S_CP) = Standby & Eval(isMissionSelected) = FALSE & S_CP recieves EvtRequestLoginHmi FROM HMI] **S** =[S_CP send EvtAckHmi WITH result=TRUE to **ALL** HMI]
027	If the S_CP is in active State and if the selected mission is CAP or CDS, upon reception of an EvtRequestLoginHmi message from HMI, the S_CP shall validate the password associated to the HMI	If **STATE**(S_CP) = active, **AND** isMissionSelected = TRUE **AND** Mission **IN** (CAP, CDS) **AND** S_CP **recieves** EvtRequestLoginHmi **FROM** HMI, **THEN** S_CP **send** validateUsername **WITH** to **ALL** HMI.	It is always the case that If **P** holds THEN **S** eventually holds. **With: P** =[STATE(S_CP) = active & Eval(isMissionSelected) = TRUE (& Eval(Mission) = CAP **OR** Eval(Mission) = CDS) & S_CP recieves EvtRequestLoginHmi FROM HMI] **S** =[S_CP send validateUsername to **ALL** HMI]
458	Upon reception of an EvtRequestUserPasswordChangeHmi, the S_CP shall change the password associated to the user and send an EvtAckHmi with result to TRUE to the HMI.	If S_CP **recieves** EvtRequestUserPasswordChangeHmi **FROM** HMI, **THEN** S_CP **send** changePasswordReq **WITH** to **ALL** HMI **AND** S_CP **send** EvtAckHmi **WITH** to **ALL** HMI **WITH** result = TRUE.	It is always the case that If **P** holds THEN **S** eventually holds. **With: P** = S_CP recieves EvtRequestUserPasswordChangeHmi FROM HMI] **S** =[S_CP send changePasswordReq to **ALL** HMI & S_CP send EvtAckHmi to ALL HMI with result = TRUE]

Table 9.1 : Exigences sur le composant SM du cas d'étude AFS 3/3 (continued)

ID	Exigence de départ	Exigences URM	Exigence consolidées
012	Upon reception of a valid EvtRequestLoginHmi from an HMI, the S_CP shall send an EvtCommandStatusHmi message with commandId to LOGOUT and status to TRUE to the logged HMI	If S_CP **recieves** EvtRequestLoginHmi **FROM** HMI, **THEN** S_CP **send** EvtCommandStatusHmi **WITH** commandId = LOGOUT **AND** status = TRUE to **ALL** HMI	It is always the case that If **P** holds THEN **S** eventually holds. **With: P** = S_CP recieves EvtRequestLoginHmi FROM HMI] **S** =[S_CP send EvtCommandStatusHmi WITH commandId = LOGOUT & status = TRUE to **ALL** HMI]
443	Upon reception of a valid EvtRequestLogoutHmi or upon reception of EvtRequestStopMissionHmi, the S_CP shall send an EvtCommandStatusHmi with commandId to LOGOUT and status to FALSE to the logged HMI.	If S_CP **recieves** EvtRequestLogoutHmi **OR** EvtRequestStopMissionHmi **FROM** HMI, **THEN** S_CP **send** EvtCommandStatusHmi **WITH** commandId = LOGOUT **AND** status = TRUE to **ALL** HMI	It is always the case that If **P** holds THEN **S** eventually holds. **With: P** = S_CP recieves EvtRequestLoginHmi FROM HMI] & S_CP recieves EvtRequestStopMissionHmi FROM HMI] **S** =[S_CP send EvtCommandStatusHmi WITH commandId = LOGOUT & status = TRUE to **ALL** HMI]
016	Upon reception of an EvtRequestMissionHmi message from an HMI, the S_CP shall send to all logged HMI an EvtAvailableRolesHmi.	If S_CP **recieves** EvtRequestMissionHmi, **THEN** S_CP **send** EvtAvailableRolesHmi	It is always the case that If **P** holds THEN **S** eventually holds. **With: P** = S_CP recieves EvtRequestMissionHmi FROM HMI] **S** =[S_CP send EvtAvailableRolesHmi to **ALL** HMI]
013	Upon reception of a valid EvtRequestLogoutHmi message from an HMI, If there is other logged operators, the S_CP shall release the operator role associated with the HMI, reallocate his role to the other logged HMI by sending a EvtCurrentRoleHmi.	If S_CP **recieves** EvtRequestLogoutHmi **AND** LoggedOperatorNumber > 0, **THEN** S_CP **send** releaseRole to HMI **AND** S_CP **send** EvtCurrentRoleHmi to HMI	It is always the case that If **P** holds THEN **S** eventually holds. **With: P** = S_CP recieves EvtRequestLogoutHmi FROM HMI] & LoggedOperatorNumber > 0 **S** =[S_CP send releaseRole to HMI] & S_CP send EvtCurrentRoleHmi to HMI

9.3 Discussion et synthèse

Nous avons présenté une application de la méthodologie de vérification proposée dans ce mémoire sur un cas d'étude industriel de taille et réelle. Tout d'abord, nous avons formalisé les interactions entre le composant à valider et son environnement. La technique proposée est basée sur la formalisation des cas d'utilisation dans des modèles CDL. Ceux-ci décrivent les interactions entre le modèle à valider et son environnement. L'hypothèse forte de cette méthode est que le concepteur est en mesure de spécifier les interactions importantes entre son système et l'environnement dans lequel le système va opérer. Nous justifions cette hypothèse, en particulier dans le domaine de l'embarqué par le fait que le concepteur d'un système doit connaître le périmètre (contraintes, conditions) de son utilisation pour pouvoir le développer correctement. En conséquence, le processus de développement doit inclure une étape de spécification de l'environnement permettant d'identifier un ensemble complet de toutes les interactions entre l'environnement et le modèle, ce qui assurait un taux de couverture de 100%. Dans notre approche, nous avons identifié une activité dédiée à la description des cas d'utilisation étendus afin d'y intégrer la spécification des scénarios d'interaction entre le système et son environnement. Le langage XUC, proposé à cet effet, intègre les constructions nécessaires permettant aux concepteurs de décrire des scénarios d'interactions simples, puis les composer sous la forme de scénarios complexes. Par la suite, cet ensemble de scénarios est traduit en une spécification formelle sous la forme de modèles CDL. De même pour les exigences des cahiers des charges. Celles-ci ont été réécrites en langage URM afin de générer des propriétés CDL. Ces propriétés sont obtenues suite à l'identification des patrons de spécification après l'analyse structurelle des exigences consolidées avec les fragments de propriétés. Ces programmes CDL sont exploités dans un processus de vérification encadré et outillé.