UNIVERSITÉ DE NANTES

Master ALMA
2015-2016

Rapport du module d'initiation à la recherche

Modélisation et analyse des systèmes logiciels hétérogènes

Auteurs :
Thibault Béziers la Fosse
Ugo Mahey
Joachim Clayton

Encadré par :
M^r Christian Attiogbé
Equipe AeLoS

29 avril 2016

Remerciements

Grâce à ce stage au sein de l'équipe AeLoS nous avons eu l'opportunité de découvrir le milieu de la recherche, et avons dû faire face à un réel problème, nous immergeant dans des conditions réelles de recherche.
Nous souhaiterions donc adresser nos remerciements à l'équipe AeLoS et M^r Attiogbé pour leur accueil et l'encadrement dont nous avons bénéficié.

Table des matières

Résumé

Aujourd'hui en informatique, chaque entreprise, chaque développeur ont des affinités plus ou moins fortes pour un ou plusieurs langages de programmation pour réaliser des applications complexes. Cette diversité peut amener des problèmes pour l'analyse la maintenabilité, la réalisation de projets ou encore l'ajout de nouvelles fonctionnalités à une entité déjà existante.
On peut se demander comment peut être mis en relation des composants informatiques dans un système hétérogène. Nous avons mené nos recherches sur les langages de modélisation SysML et Uppaal afin de vérifier la possibilité d'une mise en relation.
Nos résultats nous ont montré que cela était possible pour la majeure partie des diagrammes SysML et donc entre deux langages de modélisation. Il faut donc ainsi continuer à mener des recherches à ce sujet et développer ces idées pour peu à peu arriver à pouvoir combiner ces composants de façon naturelle.

1 Introduction

Dans le cadre du module d'initiation à la recherche du Master 1 ALMA nous avons été amené à effectuer un stage avec un enseignant du LINA dans l'équipe AeLoS. Lors de ce stage nous avons étudié les systèmes hétérogènes.

1.1 LINA

Le *LINA* est le Laboratoire d'informatique de Nantes Atlantique. Il réunit au sein de neuf équipes 70 enseignants-chercheurs, 65 doctorants, un chercheur et des post-doctorants. Il est divisé en deux groupes,

- ALD : Architectures logicielles distribuées.
  - AeLoS : Architecture et logiciels sûrs
  - Ascola : Aspect and Composition languages
  - GDD : Gestion de données distribuées
  - AtlanMOD : Atlantis Modeling
- SAD : Systèmes d'aide à la décision
  - DUKe : Data, User, Knowledge
  - ComBi : Combinatoire et BioInformatique
  - TALN : Traitement Automatique du Langage Naturel
  - Contraintes
  - OPTImization : Méthodes Ensemblistes et Optimisation, Recherche Opérationnelle et Optimisation Multicritère.

Ce laboratoire est situé sur deux sites au sein de la ville de Nantes : La Lombarderie (UFR Sciences et Techniques) et La Chantrerie (Polytech'Nantes et l'École des mines).

1.2 AeLoS

L'équipe AeLoS rassemble neuf enseignants-chercheurs permanents et une dizaine de doctorants, en plus d'assistants et d'associés.

Ses thématiques développées sont les suivantes :

- Styles de conception et d'évolution centrées architectures ;
- Spécification et vérification de composants logiciels ;
- Multiformalisme et analyse multifacette ;
- Approches formelles des systèmes embarqués communicants.

1.3 Contexte

Lors de la conception de système de taille importante, il n'est pas rare que les différents composants de ce système ne soient pas construits de la même manière : différents langages, différentes architectures... Comment faire pour que les différents composants puissent communiquer ; si l'on veut intégrer un nouveau composant au système dans quel langage l'écrire et comment l'intégrer au reste du système tout en gardant une cohérence au sein de celui-ci.

Dans ce rapport nous allons expliciter clairement les différents problèmes liés à cette hétérogénéité. Notre rapport va également reprendre en détail les différents points que nous avons explorés pour répondre à cette problématique. Nous décrirons avec précision les détails de notre solution, mais aussi montrerons divers exemples attestant de sa viabilité. Enfin nous montrerons les améliorations que nous pourrions faire, ainsi que d'autres solutions alternatives.

2 Sujet de Recherche

2.1 Problématique

La problématique principale est la suivante : comment réussir à faire communiquer différents composants hétérogènes entre eux. Les différents composants ayant des sémantiques différentes, l'assemblage de ces composants n'est pas chose aisée. Il faut que ces composants interagissent entre eux tout en gardant le système cohérent. Sans connaissance de la sémantique d'un composant voisin, les différents composants ne peuvent pas communiquer directement entre eux.

Un exemple de notre vie de tous les jours est celui du téléphone. N'importe quel téléphone peut appeler n'importe qui, peu importe le modèle de téléphone qu'il possède, son opérateur, son pays. Certains véhicules sont même capables de communiquer par le biais de ces téléphones. Cela signifie que bien que l'architecture des téléphones diffère selon leurs modèles ou leurs fabricants, ils sont capables d'homogénéiser leurs signaux afin d'être compris et de comprendre des systèmes différents du leur.

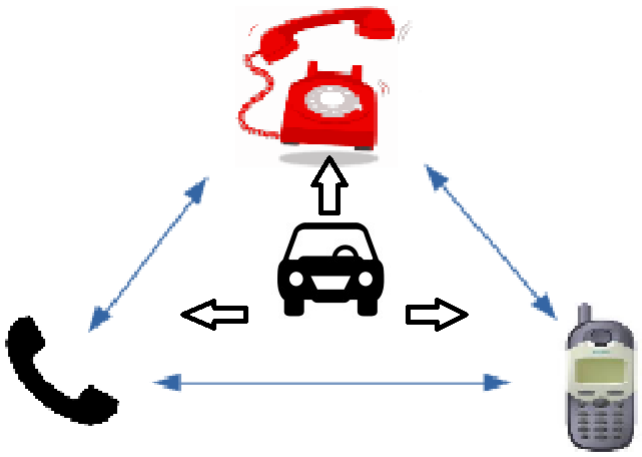


FIGURE 1 – Un système hétérogène

2.2 Système choisi

Afin de mettre en situation un système hétérogène, nous nous sommes intéressés en suivant les conseils de M^r Attiogbé à des outils de modélisation d'automates, notamment UPPAAL et SysML.

UPPAAL permet la représentation d'automates, leur validation et leur vérification. C'est un outil léger, et simple d'utilisation, qui correspond tout à fait à un sujet de recherche comme le nôtre.

SysML est un langage de modélisation spécifique à l'ingénierie système. Il offre énormément de possibilité : la spécification, l'analyse, la conception de systèmes, et bien plus encore, par le biais de nombreux types de diagrammes différents. Il reprend, et complète tous les diagrammes UML-2, en plus d'en ajouter de nouveaux, offrant des possibilités encore plus grandes.

SysML permet la représentation d'automates par le biais de diagrammes d'activité (*Activity Diagram*, ou *Actigram*), mais ne propose pas de vérification. Tandis que UPPAAL propose des vérifications d'automates en plus de leur modélisation. Ainsi notre objectif est de mettre en place un système hétérogène, proposant à UPPAAL de vérifier des automates générés avec des diagrammes d'Activité SysML. L'intérêt est de pouvoir utiliser la puissance de SysML,

qui propose énormément de diagrammes différents, avec les fonctions de vérification de Uppaal . On est donc face à un système hétérogène.

Afin d'utiliser SysML, nous avons choisi le logiciel Papyrus, qui se présente sous la forme d'un plug-in pour Éclipse. Effectivement Papyrus est Open-Source et facile d'utilisation, c'est pour ces deux raisons que nous avons décidé de nous en servir.

Cependant, les fichiers générés par Uppaal ne sont pas compatibles avec Papyrus, et réciproquement. Nos deux logiciels ne sont pas capables de communiquer entre eux, il nous faut trouver une solution.

2.2.1 Solutions envisageables

On distingue trois solutions possibles afin de résoudre un problème d'hétérogénéité.

La première solution est d'adapter une des parties pour qu'elle soit cohérente avec la deuxième. Les deux pièces vont alors être cohérentes entre-elles et vont pouvoir communiquer. Cependant cette solution implique de pouvoir modifier une des parties du système.

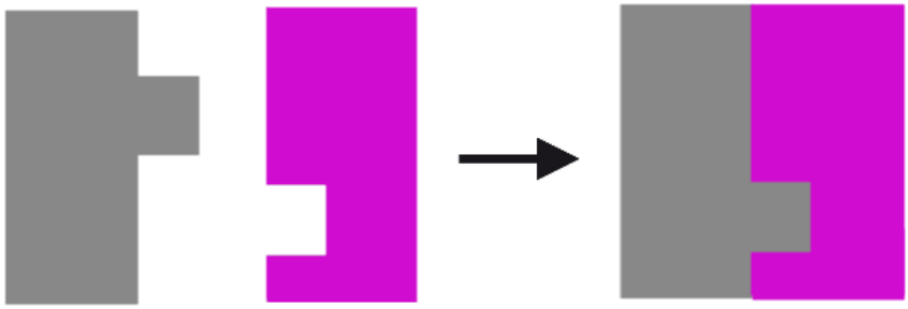


Figure 2 – Modification d'une des parties du système pour qu'elle soit compatible

Une deuxième solution est de créer artificiellement un troisième composant qui fait le lien entre les deux autre. C'est-à-dire un composant qui connaît la sémantique des deux éléments communiquants, et qui puisse ainsi modifier leurs sorties, afin qu'elles soient compatibles avec les entrées opposées.

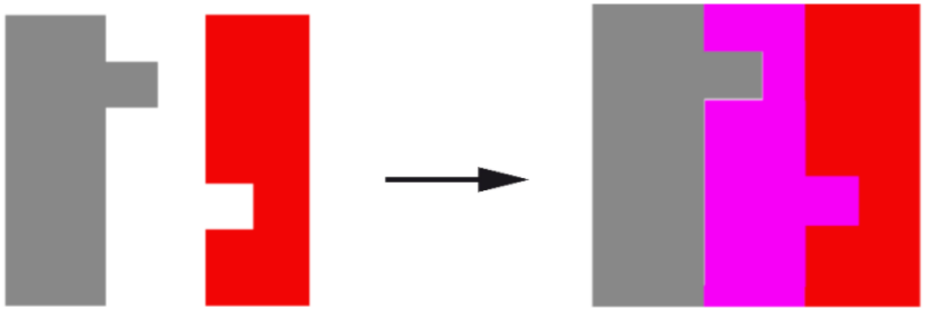


FIGURE 3 – Création d'un troisième composant pouvant harmoniser les deux autres.

Une troisième solution serait de faire communiquer "de force" ces différents composants et de voir le degré de compatibilité entre ces deux composants, par exemple en détectant les anomalies de communication.

FIGURE 4 – Mise en communication, et détection des anomalies.

2.3 Rappel sur les automates

Dans notre projet nous avons utilisé des automates à états finis non déterministes. Voici un bref rappel à ce sujet.

Définition :

1. Un ensemble d'états finis noté $\mathcal{S}$.
2. Un ensemble fini, non vide de lettres qui est l'alphabet d'entrée noté Σ.
3. Un ensemble d'états initiaux noté $\mathcal{I}$.
4. Un ensemble d'états finaux noté $\mathcal{F}$.
5. Une fonction de transition, noté δ, qui reçoit comme arguments un état de $\mathcal{S}$ et un symbole de Σ et qui rend comme résultat une partie de $\mathcal{S}$.

Voici un automate classique réutilisé un peu plus loin dans notre projet :

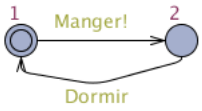


FIGURE 5 – Automate du cycle de vie d'un hamster

Il est plutôt simple : L'animal mange, dort, et recommence indéfiniment. Pour revenir à la définition formelle, on a donc :

1. Un ensemble d'états finis $\mathcal{S} = \{1, 2\}$
2. Un alphabet $\Sigma = \{Manger!, Dormir!\}$
3. Un état initial $\mathcal{I} = \{1\}$
4. Un ensemble d'états finaux : $\mathcal{F} = \emptyset$
5. Et enfin des fonctions de transition :
   (a) $\delta(1, \text{manger}\,!) = 2$
   (b) $\delta(2, \text{dormir}\,!) = 3$

Pour la suite il arrivera que les états n'aient pas forcément de noms "explicites", de même pour les transitions. Les langages de modélisation SysML et UPPAAL leur donnent un identifiant unique, si bien qu'on peut les utiliser comme alphabet, et ensemble d'états finis.

2.4 Uppaal

UPPAAL permet la modélisation d'automate, ainsi que la vérification et la simulation de leur exécution. Son interface graphique est très simple d'utilisation. UPPAAL possède trois parties principales :

— Un langage de description
— Un simulateur
— Un vérificateur de modèle

Le langage de description est un langage de commandes non déterministes, utilisant des variables avec un typage fort (Booléens, entiers, tableaux ...). Il utilise un langage de modélisation pour décrire le comportement de systèmes sous la forme de réseaux d'automates.

Le simulateur est un outil de validation permettant l'examination de possibles exécutions du système au cours de la modélisation. Il permet aussi une méthode peu coûteuse de détection d'erreurs, antécédent au vérificateur de modèle, couvrant lui l'intégralité du modèle.

Le vérificateur de modèle peut vérifier les invariants, ainsi que les propriétés en explorant l'ensemble des états du système.

Ses données sont enregistrées avec le standard XML.

2.5 SysML

Systeme Modeling Language ou SysML est un langage de modélisation. Il peut être également assimilé à une extension d'UML. Il permet essentiellement la spécification, l'analyse, la conception la vérification et la validation de systèmes complexes et/ou hétérogènes. SysML aborde la conception des systèmes avec la notion de bloc. Ceux-ci correspondront à des parties mécaniques, électroniques, informatiques ou autres à la fin du développement.

Les diagrammes SysML sont catégorisés comme suit :

— Les diagrammes structurels ou statiques
— Les diagrammes comportementaux
— Les diagrammes transversaux

Les diagrammes structurels regroupent :

— Les diagrammes de "packages"
— Les diagrammes de blocs internes
— Les diagrammes de blocs "block diagrammes" (Bdd)
— Les diagrammes paramétriques
— Les ports

Les diagrammes comportementaux eux regroupent :

— Les diagrammes de séquences
— Les diagrammes d'activité
— Les diagrammes états-transitions

Les diagrammes transversaux correspondent quant à eux aux diagrammes de spécifications.

Dans notre étude nous nous sommes concentrés sur les diagrammes comportementaux, et notamment celui d'activités. Les diagrammes d'activités permettent d'écrire sous forme de flux ou d'enchaînement d'activités le comportement du système ou de ses composants.

2.6 Extensible Markup Language (XML)

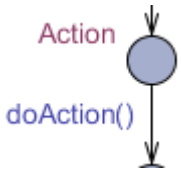
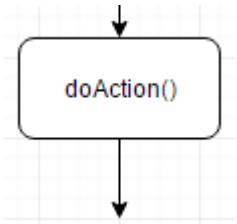
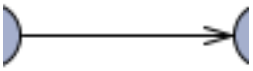
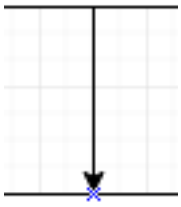
C'est un langage de balisage extensible. Cette syntaxe est dite « extensible » car elle permet de définir différents espaces de noms, c'est-à-dire des langages avec chacun leur vocabulaire et leur grammaire. Elle est reconnaissable par son usage des chevrons encadrant les balises. L'objectif de ce

langage est de faciliter l'échange automatisé de contenus complexes (arbres, texte riche...) entre systèmes d'informations hétérogènes.

3 HétéroSysML : Des *Actigram* sous Uppaal

3.1 Un rapprochement de sémantique

La première chose frappante entre des automates Uppaal et des diagrammes d'Activity SysML est la proximité de leurs modèles. Les deux sont sous la forme de graphe, avec des transitions orientées. On peut dresser le tableau suivant comparant respectivement la plupart des actions SysML avec une possible transition vers Uppaal .

Sémantique	Uppaal	SysML	Remarques
Action			Représente un pas dans l'Activity, et un état dans l'automate. Si on veut juste modéliser, on nomme le noeud Uppaal comme l'action SysML, et si on veut faire une exécution avec Uppaal on peut définir une fonction exécutée au passage du noeud.
Control Flow			Démarre une action juste après que la précédente soit effectuée. La sémantique est la même pour les langages.

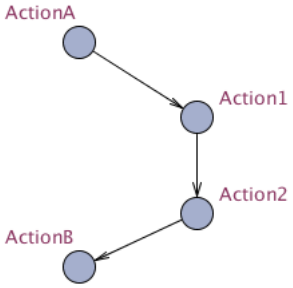
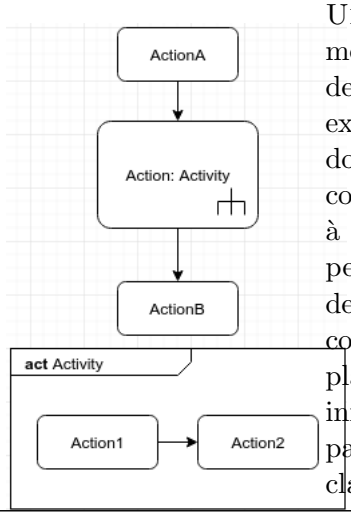
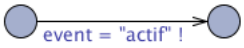
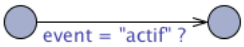
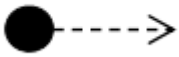
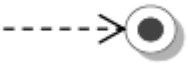
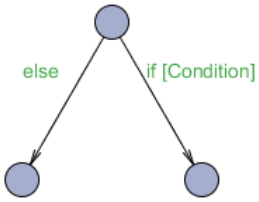
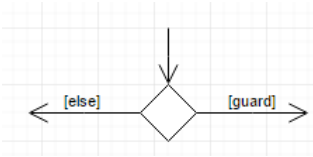
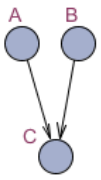
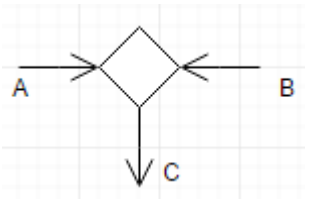
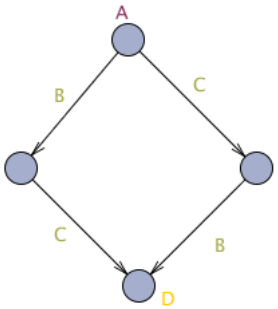
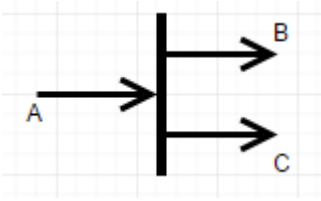
Appel d'Activity	 		UPPAAL ne permet pas l'appel de d'automates externes : on insère donc l'automate correspondant à l'Activity appelée à l'intérieur de l'automate courant, en remplaçant ses états initiaux et finaux par des états classiques.
Signal			Le signal envoyé par SysML correspond à la mise à jour d'une variable à une valeur déclenchant un état.
Event			Un event démarre quand on a reçu le signal correspondant. Il en va de même avec Uppal, on vérifie que la variable possède la valeur attendue pour démarrer l'event.

TABLE 1: Actions UPPAAL et SysML

Les actions classiques de SysML peuvent donc trouver un parallélisme vers UPPAAL . Penchons-nous ensuite sur les flux de SysML, avec le tableau suivant :

Sémantique	Uppaal	SysML	Remarques
Noeud initial			Début de l'automate, et du *Routing Flow*
Noeud Final			UPPAAL n'offre pas la possibilité de faire des Noeuds Finaux. Pour le repérer parmis les autres on nomme son état "Stop".
Décision			Choix du flux dépendant de la condition. Les conditions sont ainsi évaluées avec l'utilisation de Guard avec UPPAAL .
Merge			Avec SysML le premier flux à arriver entre A et B détermine les input de C. Avec UPPAAL il y a deux branches issues d'une condition A et B, celle qui a validé la condition en amont sera suivie par C.
Fork			UPPAAL ne permet pas le parallélisme. On doit ainsi adopter cette solution non déterminisme équivalente à une exécution parallèle d'instructions.

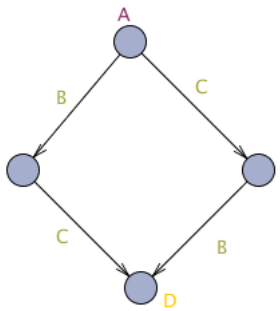
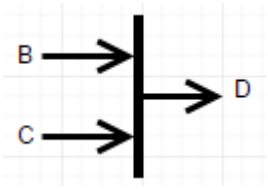
Join	 	De la même manière que le cas précédent, on ne peux pas paralléliser avec UPPAAL. On doit donc représenter ainsi le join de SysML. Notons qu'en SysMl un join indique que chaque branche doit avoir terminé sa tache avant de continuer, ce qui correspond en quelque sorte à notre représentation Uppaal .

Table 2: Routing Flows Uppaal et SysML

3.2 Liaison entre les deux modèles

Comme vue précédemment il y a trois solutions possibles pour faire communiquer deux langages (de modélisation dans notre cas) :

1. Modification d'un des deux systèmes (Uppaal ou SysML)
2. Création d'un système intermédiaire
3. Forçage de composition et détection des défauts de communication

Les deux logiciels produisant du code XML, nous avons décidé de mettre en oeuvre la seconde solution.

Nous avons choisi de créer un composant à deux interfaces, une interface pour les automates et une interface pour le SysML. Notre objectif est de vérifier un modèle SysML avec Uppaal .

Comme dit précédemment nous allons utiliser un troisième composant intermédiaire que nous avons nommé HeteroSysML. Les deux supports utilisés (Uppaal et Papyrus) sont des logiciels de modélisation des différents diagrammes et schémas. Ces deux logiciels utilisent le XML pour représenter les différents éléments du diagramme et du schéma. Nous avons donc commencé par repérer les rapprochements au niveau du code XML entre les deux logiciels.

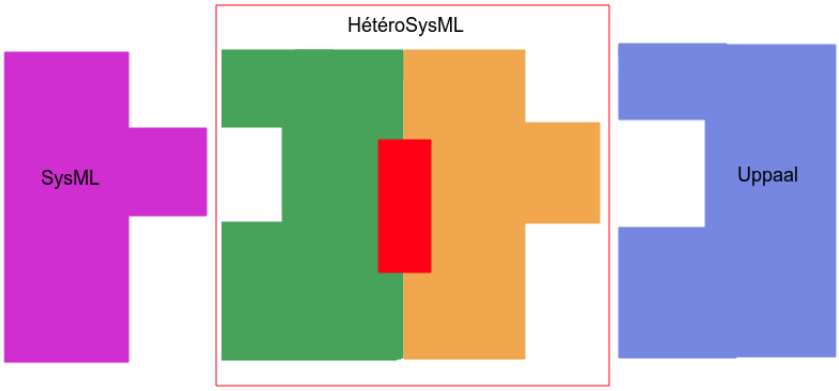


FIGURE 6 – Principe de HétéroSysML

Chaque logiciel a sa propre méthode pour enregistrer ses données. Notre objectif est donc d'écrire un programme capable de parcourir un code XML pour en extraire les informations intéressantes, et générer le code XML correspondant à l'autre logiciel.

Après avoir vu les correspondances sémantiques il faut maintenant voir les correspondances syntaxiques entre l'XML d'Uppaal et celui de Papyrus.

Analysons les différentes formes d'XML présentes dans Uppaal et dans Papyrus.

3.2.1 XML pour Uppaal

Analysons ce code XML : Chaque noeud est dans une balise location qui contient ses coordonnées dans UPPAAL . Le nom est aussi placé dedans avec ses coordonnées.

Les liens sont placés dans les balises transition, la "source" indique l'identifiant du noeud de départ, et la "ref" l'identifiant du noeud d'arrivée. Lorsqu'on a kind="guard", c'est que le label de la transition correspond à une entrée ou une sortie, et lorsque le label est "select" c'est qu'on a à faire à une condition.

En partant de ça on peut reconstruire un graphe d'activity SysML, pour peu qu'on puisse lire le format de sauvegarde de la même manière.

Cependant on peut déjà imaginer une petite application prenant un format UPPAAL en entrée et sortant du SysML sous la forme de diagramme d'activity. Un simple *parser* d'XML ferait l'affaire.

Un exemple de code XML brut est en annexe A.

3.2.2 XML pour Papyrus

On se rappelle qu'un automate représente des états avec des liaisons entre chaque et SysML représente différentes actions avec un ou plusieurs Input(s) et un ou plusieurs Output(s).
On peut à nouveau décrypter ce code. Les nodes correspondent aux locations avec Uppaal , avec en plus la possibilité d'avoir plusieurs types différents. (DecisionNode, OpaqueAction, OutputPin etc...) De plus les transitions sont ici aussi présentes, néanmoins elles sont nommées "Edge". Elles partent d'une ID et se rendent vers une autre ID, exactement comme avec Uppaal , si ce n'est que les ID de nos objets sont comme hashées. Mais elles commencent toutes par un "_".

Néanmoins nous n'avons pas la fonction de hashage de Papyrus, et nous ne connaissons pas la manière avec laquelle l'identifiant est généré à partir de l'abscisse et de l'ordonnée de chaque action. Nous partons du principe qu'elle est trouvable dans les sources de Papyrus, qui est Open-Source.

Enfin on peut se pencher un peu plus vers le decision node, représentant les conditions dans ce diagramme d'activity. Il possède un noeud initial appelé incoming="..._id du node initial" et deux noeuds de sorties : outgoing="..._id1_id2".

Un exemple de code XML brut est en annexe B.

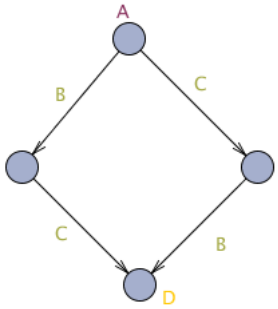
3.2.3 XML Papyrus à XML Uppaal

Avec les différentes analyses faites sur les fichiers XML de Uppaal et ceux de Papyrus il est alors possible d'effectuer la conversion de l'un vers l'autre.

Voici plusieurs conversions possibles de patterns courants.

Sémantique	Uppaal	SysML
Noeud	<location id="id0" x="-216" y="-136"></location>	<node xmi :type="uml :Node" xmi :id= hash(id0, -216, -136) />
	On garde l'ID, et la position x,y qui permettent d'obtenir le code exact pour chacun des langages. Notons qu'avec SysML toutes ces données sont à l'intérieur du champs xmi :id.	
Noeud Initial	<location id="id0" x="-216" y="-136"></location><init ref="id0"/>	<node xmi :type="uml :InitialNode" xmi :id = hash(id0, -216, -136)/>

<table>
<tr><td></td><td colspan="2">Uppaal ajoute un champs init définissant le noeud initial à partir de son id, tandis que SysML modifie le type du noeud. L'algorithme ajouterait donc ce champs init dans un sens, ou bien donne le type InitialNode dans l'autre sens.</td></tr>
<tr><td>Noeud Final</td><td><location id="id0" x="-216" y="-136"><name x="-206" y="-103" >stop </name></location></td><td><node xmi :type="uml :ActivityFinalNode" xmi :id = hash(id0, -216, -136)/></td></tr>
<tr><td></td><td colspan="2">L'algorithme reste simple : Ajout d'un champs name avec pour valeur stop aux coordonnées $x + 10$, $y - 30$, ou bien utilisation du type ActivityFinalNode vers SysML.</td></tr>
<tr><td>Flow Control</td><td><transition><source ref="id2"/><target ref="id1"/></transition></td><td><edge xmi :type="uml :ControlFlow" xmi :id= hash(id1, id2) target="_id1" source="_id2"/></td></tr>
<tr><td></td><td colspan="2">Les arêtes sous Uppaal n'ont pas d'id alors que sous SysML elles en possèdent une. On suppose qu'elle est issue d'un hash entre le noeud de source et de destination, que l'on peut générer dans notre algorithme. Autrement, c'est plutôt transparent : On a une balise transition d'un côté, edge de l'autre. Là où Uppaal utilise des balises pour définir les sources et destinations, SysML utilise des attributs à l'intérieur de sa balise edge.</td></tr>
<tr><td>Conditions</td><td><transition><source ref="id2"/><target ref="id1"/><label kind="guard">if [Condition] </label></transition> <transition><source ref="id2"/><target ref="id3"/><label kind="guard">else </label></transition></td><td><node xmi :type="uml :DecisionNode" xmi :id="_hash(x, y, id)" name="DecisionNode1" incoming="_id2" outgoing="_id1 _id3"/></td></tr>
<tr><td></td><td colspan="2">Ici on a une différence majeure : Uppaal définit les conditions comme deux arcs avec un label, alors que SysML définit un noeud particulier.</td></tr>
</table>

<table>
<tr><td>Merge</td><td><location id=”id0” x=”-120” y=”-120”></location>
<location id=”id1” x=”-184” y=”-112”></location>
<location id=”id2” x=”-144” y=”-40”></location>
<transition><source ref=”id2”/><target ref=”id0”/></transition>
<transition><source ref=”id2”/><target ref=”id1”/></transition></td><td>xmi :type=”uml :MergeNode” xmi :id=”_id” name=”MergeNode1” incoming=”_id1 _id2” outgoing=”_id3”/></td></tr>
<tr><td colspan="2">On retrouve ici aussi la différence énoncée pour la conditionnelle. Uppaal lie simplement deux noeuds à un même troisième noeud, là ou Sysml possède un noeud particulier dédié à cet effet. Uppaal modélisant des automates, il n’y a pas d’exécution simultanée. Si bien que le Merge node de SysML, permettant au premier flux arrivant de prendre le contrôle du reste n’a pas de sens avec Uppaal et ses automates.</td></tr>
<tr><td>Fork</td><td> 
</td><td><node xmi :type=”uml :ForkNode” xmi :id=”_id” name=”ForkNode1” incoming=”_id1” outgoing=”_id2, _id3”/></td></tr>
<tr><td colspan="2">Comme expliqué précédemment, le fork n’est pas disponible dans Uppaal . Si bien que nous faisons autant de branche que d’arrangements possibles de noeuds, qui ne seront pas écrit ici pour une raison de lourdeur du code XML.</td></tr>
</table>

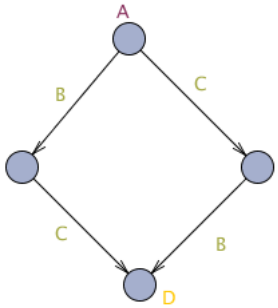
Join	 Le Join est à la suite du fork node. Ainsi si ce dernier n'est pas représentable sur Uppaal , le join node non plus.	<node xmi :type="uml :JoinNode" xmi :id="_id" name="JoinNode1" incoming="_id1, _id2" outgoing="_id3" />

Table 3: Grammaires

3.3 Structure de donnée, et pseudo-algorithme

Les diagrammes SysML de Papyrus, et les automates Uppaal sont enregistrés avec le même format : XML. Le XML est un langage extrêmement facile à parcourir (à *Parser*), si bien que ce sera vraiment aisé de récupérer toutes les informations dont nous avons besoin pour faire fonctionner notre système en symbiose.

De cette manière on peut imaginer comment nous pourrions stocker les informations lues pour que le passage d'Uppaal à SysML, et réciproquement, dans un hypothétique programme. Nous utiliserions l'arborescence de classe suivante afin d'organiser nos données :

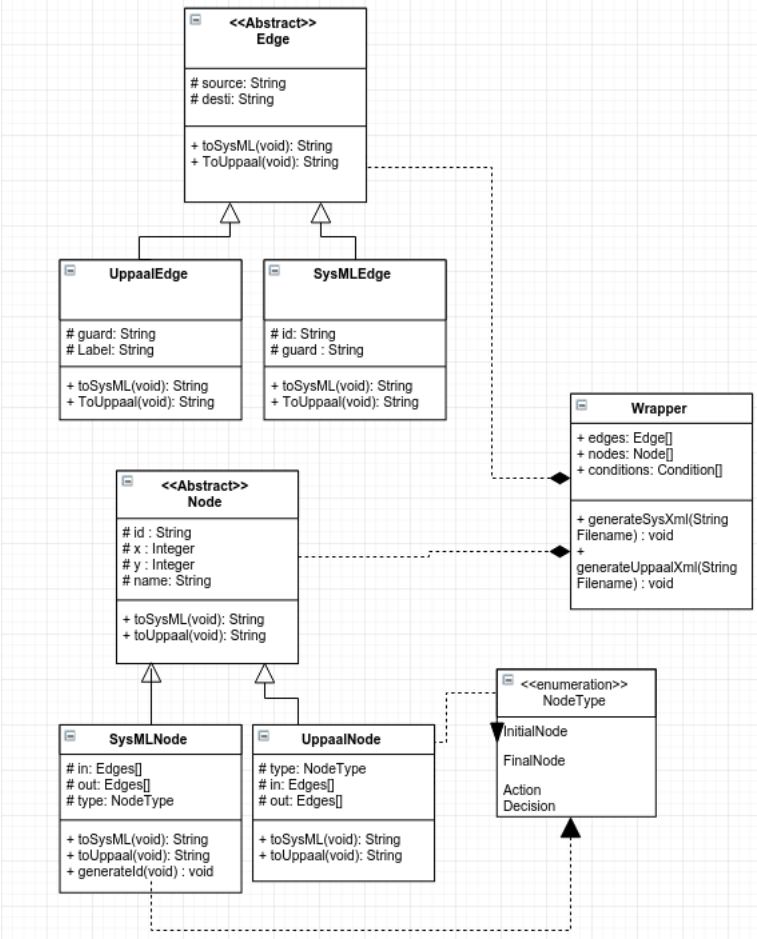


FIGURE 7 – Architecture de HétéroSyML

Chaque classe définit donc un type d'objet "Parallèle" entre SysML et UPPAAL :

— Les noeuds UPPAAL et les Actions SysML
— Les transitions UPPAAL et les Arêtes SysML
— Les conditions UPPAAL et les noeuds de décision SysML
— (etc)

Les classes abstraites définissent ce type, et les classes concrètes en héritant définissent l'objet issu de Papyrus ou de UPPAAL . Par exemple si on étudie un automate UPPAAL, notre programme va générer des objets de types UppaalNode, UppaalEdge, etc ... Et réciproquement avec un code issu de Papyrus.

Ces classes concrètes possèdent toutes une méthode *ToSysml()* et *toUppaal()* générant le code correspondant en XML, pour l'un ou l'autre.

Le point fort de cette architecture est la classe *Wrapper*, qui va générer le code complet indépendamment des objets qu'elle contient. On peut approximer une complexité en $O(n * m)$, avec n le nombre de noeuds, et m le nombre d'arêtes. Effectivement, pour chaque noeud parcouru, on doit parcourir l'ensemble des arêtes afin de trouver ses arêtes entrantes et sortantes.

Puisque nous sommes dans le sens SysML $\rightarrow$ UPPAAL nous pourrons "tager" les noeuds posant problèmes (fork, join) avant de mettre en place les solutions proposées précédemment avec UPPAAL. Ainsi la reconstruction du SysML après vérification pourra prendre en compte ces noeuds, non présents dans un automate, et réécrire le code XML correspondant.

Néanmoins l'ajout de noeuds depuis UPPAAL, reprenant la sémantique de ces noeuds en question ne pourra jamais générer ces noeuds spéciaux. Le sens UPPAAL $\rightarrow$ SysML n'est pas tout à fait au point encore.

Notons que l'arborescence de classe ci-dessus ne permet pas l'utilisation de toutes les possibilités d'un *Actigram*. Nous n'avons représenté que les principales fonctions.

Il existe de nombreux parser efficaces pour extraire des informations stockées dans du XML, *lxml* du langage *Python* par exemple, qui est très facile d'utilisation.

3.3.1 Concrétisation de HétéroSysML

Pour avoir un exemple concret nous utiliserons le simple modèle SysML issu de Papyrus suivant :

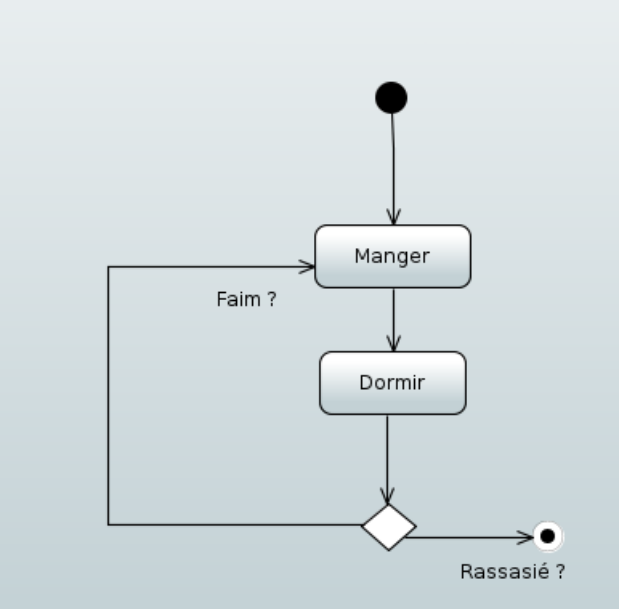


FIGURE 8 – Cycle de vie d'un hamster avec Papyrus

Le programme Python en annexe C permet de récupérer les noeuds les plus simples, ainsi que leurs arêtes correspondantes. Évidemment c'est un prototype, le programme final serait bien plus complet. Ici on ne traite que les noeuds simples, ainsi qu'une conditionnelle.

En l'exécutant sur le fichier XML généré par Papyrus on obtient les résultats en annexe D.

Ces éléments sont ensuite stockés dans les classes SyMLNode, SysMLEdge, SysMLCondition, elles-mêmes dans le Wrapper, de cette façon :

Node	Id	Nom	In	Out	Type
	id0			e0	InitialNode
	id1	Manger	e0, e2	e1	Action
	id2	Dormir	e1	e3	Action
	id3		e3	e2, e4	DecisionNode
	id4		e4		FinalNode

Edges	Id	Nom	Source	Destination
	e0		id0	id1
	e1		id1	id2
	e2	Faim?	id3	id1
	e3		id2	id3
	e4	Rassasié?	id3	id4

FIGURE 9 – Objets générés par HétéroSysML après *Parsing* du code XML

Enfin l'exécution de la méthode GenerateUpXml du Wrapper va générer les Headers XML correspondant à un fichier Uppaal, et appeler tour à tour les méthodes toUppaal() de chaque nodes, edges, ou condition.

Cette méthode va inscrire dans le fichier le code XML correspondant, en fonction de ses paramètres, à l'affichage de l'objet en question dans Papyrus.

Plus spécifiquement l'algorithme est le suivant :

1. Parsing du code XML de UPPAAL
2. Génération des objets de type sysmlNode, sysmlEdge
3. Ecriture des headers XML correspondants à un fichier UPPAAL
4. Appel de la fonction generateSysXML() du *Wrapper*
5. GenerateSysXML appelle la méthode ToUppalXML de chaque sysML-Node et SysMLEdge généré précédemment
6. Ecriture des footers XML correspondants à un fichier UPPAAL
7. Fermeture du fichier

L'exécution du pseudo-algorithme précédent permet donc de passer du modèle SysML au modèle UPPAAL. Qui correspond, une fois ouvert dans UPPAAL au diagramme suivant :

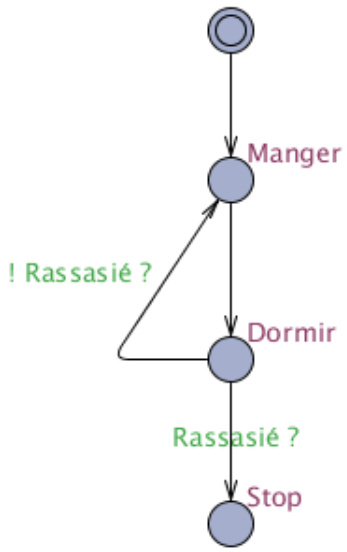


FIGURE 10 – Cycle de vie d'un hamster généré avec HétéroSysML pour UPPAAL

Nous n'avons pas implémenté l'algorithme précédent, et le résultat obtenu est issu d'une exécution *à la main*. Cependant une telle implémentation serait relativement aisée à mettre en oeuvre.

4 Conclusion

4.1 Travail effectué

Lors de ce stage d'initiation à la recherche nous avons appréhendé le problème des systèmes hétérogènes. Nous avons travaillé sur une petite instance de ce domaine, mais qui nous a néanmoins permis d'entrevoir les problèmes que cela peut poser.

Notre solution propose d'utiliser le langage XML pour homogénéiser les entrées et sorties de Papyrus avec UPPAAL. L'utilisation d'un programme placé entre les deux permet de laisser les deux logiciels agir indépendamment l'un de l'autre, mais en contrepartie requiert une constance au niveau de la syntaxe des fichiers générés par les programmes.

On suppose qu'il est correct, même si aucune preuve concrète n'aie été faite. Cependant nous aurions aimé pouvoir générer un code Papyrus à partir

du code Uppaal en plus.

Effectivement, ceci pourrait permettre à un utilisateur d'ouvrir un code Papyrus avec Uppaal, le modifier et vérifier sa validité, et enfin l'avoir à nouveau en SysML, et ainsi profiter de la grande variété de ses modèles. Ce serait une des futures améliorations possibles de HétéroSysML.

4.2 Perspectives

Les possibilités de SysML sont bien plus grandes que la simple utilisation du diagramme d'Activité. Une homogénéité entre SysML et Uppaal est sans doute aussi possible avec d'autres diagrammes de SysML : Paramétrique, Bloc, Classe ... L'idéal serait bien évidemment une prise en charge de tous les modèles par Uppaal , mais ça n'apporterait pas forcément grand chose, le but d'Uppaal n'étant pas d'afficher des diagrammes de classes.

4.3 Apports de cette initiation à la recherche

En conclusion, cette initiation a la recherche a été une expérience très intéressante. Nous avons pu nous mettre dans la peau d'un chercheur, et réfléchir à un problème qui n'a pas encore de solution. En travaillant sur une instance relativement petite comme SysML et Uppaal nous avons pu vraiment nous investir dans nos travaux, même si la gestion du temps nous a posé quelques problèmes.

Nous garderons malgré ça un souvenir positif de cette expérience.

Références

[1] Michel Goossens, Frank Mittelbach, and Alexander Samarin. *An Approach Combining Simulation and Verification for SysML using SystemC and Uppall.* 1990
`http://cal2014.enseeiht.fr/papers/An Approach Combining Simulation and Verification for SysML using SystemC and Uppaal.pdf`

[2] Antoine Forgerou, Jérémy Bardon, Nicolas Bourdin.
Modélisation et analyse des systèmes logiciels hétérogènes.
Université de Nantes - Avril 2015

[3] `https://eclipse.org/papyrus/`

[4] `http://www.omgsysml.org/`

[5] `http://www.uml-sysml.org/`

[6] `http://www.uppaal.org`

[7] `https://www.visual-paradigm.com/VPGallery/diagrams/Activity.html/`

A Code XML issu de uppaal

```
<template>
<name x="5" y="5">Template</name>
<declaration>// Place local declarations here.</declaration>
<location id="id0" x="-112" y="-24">
<name x="-122" y="-54">out2</name>
</location>
<location id="id1" x="-128" y="-136">
<name x="-136" y="-168">in2</name>
</location>
<location id="id2" x="-144" y="-272">
<name x="-154" y="-302">out1</name>
</location>
<location id="id3" x="-840" y="-272">
<name x="-850" y="-302">In1</name>
</location>
<location id="id4" x="-400" y="104">
<label kind="invariant" x="-410"y="119">Stop</label>
</location>
<location id="id5" x="-400" y="-24">
<name x="-410" y="-54">a5</name>
</location>
<location id="id6" x="-296" y="-136">
<name x="-306" y="-166">a4</name>
</location>
<location id="id7" x="-512" y="-144">
<name x="-522" y="-174">a3</name>
</location>
<location id="id8" x="-424" y="-272">
<name x="-408" y="-304">a2</name>
</location>
<location id="id9" x="-696" y="-272">
<name x="-720" y="-312">a1</name>
</location>
<location id="id10" x="-544" y="-392"></location>
<init ref="id10"/>
<transition><source ref="id5"/>
<target ref="id0"/>
<label kind="guard" x="-264" y="-48">out1</label>
```

```
</transition>
<transition>
<source ref="id1"/><target ref="id6"/>
<label kind="guard" x="-224" y="-160">in1</label>
</transition>
<transition><source ref="id8"/>
<target ref="id2"/>
<label kind="guard" x="-312" y="-296">out1</label>
</transition>
<transition>
<source ref="id3"/>
<target ref="id9"/>
<label kind="guard" x="-784" y="-296">in1</label>
</transition>
<transition><source ref="id9"/>
<target ref="id7"/>
<label kind="guard" x="-680" y="-176">in1</label>
<nail x="-696" y="-144"/>
</transition>
<transition>
<source ref="id5"/><target ref="id4"/>
</transition>
<transition>
<source ref="id6"/>
<target ref="id5"/>
<label kind="guard" x="-344" y="-72">in1</label>
</transition>
<transition>
<source ref="id7"/>
<target ref="id5"/>
<label kind="guard" x="-488" y="-80">in1</label>
</transition>
<transition>
<source ref="id8"/>
<target ref="id6"/>
<label kind="select" x="-360" y="-224">x&lt;=0</label>
</transition>
<transition>
<source ref="id8"/>
<target ref="id7"/>
```

```
<label kind="select" x="-504" y="-224">x&gt;0</label>
<label kind="comments">[x&gt;0]</label>
</transition>
<transition>
<source ref="id10"/>
<target ref="id8"/>
<nail x="-424" y="-392"/>
</transition><transition>
<source ref="id10"/>
<target ref="id9"/>
<nail x="-696" y="-392"/>
<nail x="-696" y="-384"/>
</transition>
</template><system>
```

B Code XML issu de Papyrus

```
<uml:Model xmi:version="20131001"
xmlns:xmi="http://www.omg.org/spec/XMI/20131001"
xmlns:uml="http://www.eclipse.org/uml2/5.0.0/UML"
xmi:id="_ZB-AUPv6EeWOIPCUrnmAdg" name="model">
  <packagedElement xmi:type="uml:Activity"
  xmi:id="_ZCBDoPv6EeWOIPCUrnmAdg" name="Activity1"
  node="_qAy48Pv6EeWOIPCUrnmAdg _usuckPv6EeWOIPCUrnmAdg
  _1YCeAPv6EeWOIPCUrnmAdg _buD9sPv7EeWOIPCUrnmAdg
  _1jCXoPv7EeWOIPCUrnmAdg _7erPoPv7EeWOIPCUrnmAdg
  _bApBkPv8EeWOIPCUrnmAdg _fJMLUPv8EeWOIPCUrnmAdg">
    <edge xmi:type="uml:ControlFlow"
    xmi:id="_6a0bYPv6EeWOIPCUrnmAdg"
    target="_1YCeAPv6EeWOIPCUrnmAdg"
    source="_qAy48Pv6EeWOIPCUrnmAdg"/>
    <edge xmi:type="uml:ControlFlow"
    xmi:id="_p5JjYPv7EeWOIPCUrnmAdg"
    target="_buD9sPv7EeWOIPCUrnmAdg"
    source="_1YCeAPv6EeWOIPCUrnmAdg"/>
    <edge xmi:type="uml:ControlFlow"
    xmi:id="_9z-6IPv7EeWOIPCUrnmAdg"
    target="_7erPoPv7EeWOIPCUrnmAdg"
```

```
source="_1YCeAPv6EeWOIPCUrnmAdg"/>
<edge xmi:type="uml:ControlFlow"
xmi:id="_IUfEcPv8EeWOIPCUrnmAdg"
target="_G5VVEPv8EeWOIPCUrnmAdg"
source="_EzY8QPv8EeWOIPCUrnmAdg"/>
<edge xmi:type="uml:ControlFlow"
xmi:id="_bA0nwPv8EeWOIPCUrnmAdg"
target="_bApBkPv8EeWOIPCUrnmAdg"
source="_buD9sPv7EeWOIPCUrnmAdg"/>
<edge xmi:type="uml:ControlFlow"
xmi:id="_fJQcwPv8EeWOIPCUrnmAdg"
target="_fJMLUPv8EeWOIPCUrnmAdg"
source="_bApBkPv8EeWOIPCUrnmAdg"/>
<edge xmi:type="uml:ControlFlow"
xmi:id="_xcTWUPv8EeWOIPCUrnmAdg"
target="_1jCXoPv7EeWOIPCUrnmAdg"
source="_bApBkPv8EeWOIPCUrnmAdg"/>
<node xmi:type="uml:InitialNode"
xmi:id="_qAy48Pv6EeWOIPCUrnmAdg"
name="InitialNode1" outgoing="_6a0bYPv6EeWOIPCUrnmAdg"/>
<node xmi:type="uml:ActivityFinalNode"
xmi:id="_usuckPv6EeWOIPCUrnmAdg"
name="ActivityFinalNode1"/>
<node xmi:type="uml:ForkNode"
xmi:id="_1YCeAPv6EeWOIPCUrnmAdg"
name="ForkNode2" incoming="_6a0bYPv6EeWOIPCUrnmAdg"
outgoing="_p5JjYPv7EeWOIPCUrnmAdg
_9z-6IPv7EeWOIPCUrnmAdg"/>
<node xmi:type="uml:OpaqueAction"
xmi:id="_buD9sPv7EeWOIPCUrnmAdg"
name="OpaqueAction1" incoming="_p5JjYPv7EeWOIPCUrnmAdg"
outgoing="_bA0nwPv8EeWOIPCUrnmAdg"/>
<node xmi:type="uml:OpaqueAction"
xmi:id="_1jCXoPv7EeWOIPCUrnmAdg"
name="OpaqueAction2" incoming="_xcTWUPv8EeWOIPCUrnmAdg">
  <inputValue xmi:type="uml:InputPin"
  xmi:id="_G5VVEPv8EeWOIPCUrnmAdg"
  name="InputPin1" incoming="_IUfEcPv8EeWOIPCUrnmAdg">
    <upperBound xmi:type="uml:LiteralInteger"
    xmi:id="_G5V8IPv8EeWOIPCUrnmAdg" value="1"/>
```

```
      </inputValue>
    </node>
    <node xmi:type="uml:OpaqueAction"
    xmi:id="_7erPoPv7EeWOIPCUrnmAdg"
    name="OpaqueAction3" incoming="_9z-6IPv7EeWOIPCUrnmAdg">
      <inputValue xmi:type="uml:InputPin"
      xmi:id="_A37MQPv8EeWOIPCUrnmAdg"
      name="InputPin1">
        <upperBound xmi:type="uml:LiteralInteger"
        xmi:id="_A37zUPv8EeWOIPCUrnmAdg" value="1"/>
      </inputValue>
      <outputValue xmi:type="uml:OutputPin"
      xmi:id="_EzY8QPv8EeWOIPCUrnmAdg"
      name="OutputPin1" outgoing="_IUfEcPv8EeWOIPCUrnmAdg">
        <upperBound xmi:type="uml:LiteralInteger"
        xmi:id="_EzaKYPv8EeWOIPCUrnmAdg" value="1"/>
      </outputValue>
    </node>
    <node xmi:type="uml:DecisionNode"
    xmi:id="_bApBkPv8EeWOIPCUrnmAdg"
    name="DecisionNode1" incoming="_bAOnwPv8EeWOIPCUrnmAdg"
    outgoing="_fJQcwPv8EeWOIPCUrnmAdg _xcTWUPv8EeWOIPCUrnmAdg"/>
    <node xmi:type="uml:OpaqueAction"
    xmi:id="_fJMLUPv8EeWOIPCUrnmAdg"
    name="OpaqueAction4" incoming="_fJQcwPv8EeWOIPCUrnmAdg"/>
  </packagedElement>
</uml:Model>
```

C Script Python d'extraction de données XML

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

from lxml import etree

tree = etree.parse("uppaal.xml")

for node in tree.xpath("/nta/template/location"):
```

```
        print ("Element: ")
        print ("x : "+node.get("x"))
        print ("y : "+node.get("y"))
        print ("id : "+node.get("id"))
        print ("name: "+node.find("name").text)
        print " "

    for edge in tree.xpath("/nta/template/transition"):
        print ("Arete: ")
        print ("Source: "+edge.find("source").get("ref"))
        print ("Destination: "+edge.find("target").get("ref"))
        if (edge.find("label") != None):
        print ("Condition: "+ edge.find("label").get("kind")+
            " : "+ edge.find("label").text)
```

D Sortie du script Python

```
Element:
x : -136
y : 72
id: id0
name: Stop

Element:
x : -136
y : -16
id: id1
name: Dormir

Element:
x : -136
y : -112
id: id2
name: Manger

Element:
x : -136
y : -192
id: id3
name: Init
```

```
Arete:
Source: id1
Destination: id2
Condition: guard : ! Rassasié ?
Arete:
Source: id1
Destination: id0
Condition: guard : Rassasié ?
Arete:
Source: id2
Destination: id1
Arete:
Source: id3
Destination: id2
```

E Sortie de HeteroSysML

```
<edge xmi:type="uml:ControlFlow"
xmi:id="_IUfEcPv8EeWOIPCUrnmAdg" />

<edge xmi:type="uml:ControlFlow"
xmi:id="_F70RIAuYEeaCQ7_ZkYustw"
target="_DM83IAuYEeaCQ7_ZkYustw"
source="_75U4gAuXEeaCQ7_ZkYustw" />

<edge xmi:type="uml:ControlFlow"
xmi:id="_Q_FKQAuYEeaCQ7_ZkYustw"
target="_R8N9YAuXEeaCQ7_ZkYustw"
source="_KycB0AuYEeaCQ7_ZkYustw" />

<edge xmi:type="uml:ControlFlow"
xmi:id="_SRnwMAuYEeaCQ7_ZkYustw"
name="Rassasie?"
target="__ye7YAuXEeaCQ7_ZkYustw"
source="_R8N9YAuXEeaCQ7_ZkYustw" />

<edge xmi:type="uml:ControlFlow"
```

```
xmi:id="_WTCXMAuYEeaCQ7_ZkYustw"
target="_KycB0AuYEeaCQ7_ZkYustw"
source="_DM83IAuYEeaCQ7_ZkYustw" />

<edge xmi:type="uml:ControlFlow"
xmi:id="_ZAYHMAuYEeaCQ7_ZkYustw"
name="Faim?"
target="_DM83IAuYEeaCQ7_ZkYustw"
source="_R8N9YAuXEeaCQ7_ZkYustw" />

<node xmi:type="uml:DecisionNode"
xmi:id="_R8N9YAuXEeaCQ7_ZkYustw"
name="DecisionNode1"
incoming="_Q_FKQAuYEeaCQ7_ZkYustw"
outgoing="_SRnwMAuYEeaCQ7_ZkYustw _ZAYHMAuYEeaCQ7_ZkYustw" />

<node xmi:type="uml:InitialNode"
xmi:id="_75U4gAuXEeaCQ7_ZkYustw"
name="InitialNode1"
outgoing="_F70RIAuYEeaCQ7_ZkYustw" />

<node xmi:type="uml:ActivityFinalNode"
xmi:id="__ye7YAuXEeaCQ7_ZkYustw"
name="ActivityFinalNode2"
incoming="_SRnwMAuYEeaCQ7_ZkYustw" />

<node xmi:type="uml:OpaqueAction"
xmi:id="_DM83IAuYEeaCQ7_ZkYustw"
name="Manger"
incoming="_F70RIAuYEeaCQ7_ZkYustw _ZAYHMAuYEeaCQ7_ZkYustw"
outgoing="_WTCXMAuYEeaCQ7_ZkYustw" />

<node xmi:type="uml:OpaqueAction"
xmi:id="_KycB0AuYEeaCQ7_ZkYustw"
name="Dormir"
incoming="_WTCXMAuYEeaCQ7_ZkYustw"
outgoing="_Q_FKQAuYEeaCQ7_ZkYustw" />
```