

MODÉLISATION ORIENTÉE ASPECTS DES SYSTÈMES TEMPS RÉEL

Naoufel MACHTA

M. Taha BENNANI

Samir BEN AHMED

FST / MOSIC

ISIMM

INSAT

naoufel.machta@isi.rnu.tn

taha.bennani@enit.rnu.tn

samir.benahmed@fst.rnu.tn

RÉSUMÉ : *La conception des systèmes temps réel est difficile et coûteuse. Les concepteurs temps réel doivent assurer la conception fonctionnelle du système ainsi que la définition des contraintes temps réel. La programmation par aspects permet la séparation des mécanismes fonctionnels et non-fonctionnels. Ceci permet d'améliorer la productivité et de réduire les coûts.*

Nous présentons dans cet article une approche pour la conception des applications temps réel basées sur le principe de séparation des préoccupations. L'idée de base est de « tisser » des contraintes temps réel sur une application standard. Nous détaillons notre approche avec un cas d'étude modélisé avec UML et le profil UML MARTE (i.e. UML profile for Modeling and Analyzing Real-Time and Embedded systems)

MOTS-CLÉS : *Modélisation par aspects, Application temps réel, UML, UML MARTE*

1 INTRODUCTION

La conception des systèmes temps réel est difficile et coûteuse. Les concepteurs temps réel doivent assurer la conception fonctionnelle du système et prendre en compte les contraintes temps réel. La séparation des préoccupations lors de la phase de conception peut aider à améliorer la productivité et réduire les coûts.

La programmation par aspects offre des moyens pour la description des préoccupations transverses (Kiczales et al. 1997). Elle permet la séparation des mécanismes fonctionnels des mécanismes non-fonctionnels (« secondaires »). Les premiers peuvent être un système de crédit ou de facturation. Les seconds, cependant, peuvent être des mécanismes d'authentification ou de cryptage des données.

Les implémentations actuelles de l'AOP (*Aspect Oriented Programming*) sont fondées sur le tissage à la compilation, au moment du chargement et à l'exécution. Le moment de tissage (*binding time*) mentionnées ci-dessus dépend des langages utilisés pour décrire les systèmes et leurs aspects. Par exemple, le tissage à la compilation dépend du langage de programmation, le format « .classe » est inévitable pour faire un tissage au moment du chargement et le tissage à l'exécution est basé sur le format ELF¹.

Dans ce papier nous présentons une approche générale pour la conception des systèmes temps réel embarqués basée sur la modélisation orientée aspect. Le tissage dans notre approche sera réalisé dans la phase

de modélisation (design time). En premier lieu, nous discutons les travaux connexes en relation avec l'orienté aspect à la modélisation et l'orienté aspect appliqué aux systèmes temps réel (Section 2). Par la suite, nous introduisons notre approche d'une façon générale avec la définition des concepts de l'orienté aspects que nous allons utiliser (Section 3).

Ensuite, nous présentons l'application de notre approche au domaine de modélisation orienté objet. Nous traitons le cas de modélisation avec UML et le profil UML MARTE. Nous appliquerons l'approche à un cas d'étude qui consiste à une station de commande d'un robot télé-opéré (Section 4).

2 TRAVAUX CONNEXES

Dans (Carley & Stewart 2001) les auteurs proposent un environnement de programmation orientée aspect visuelle (VAOP) pour les systèmes embarqués (*Resource-Constrained real-time Embedded Systems*). L'environnement est conçu pour le développement des systèmes temps réel embarqués à base de PBO (*Port-Based Object*) (Stewart et al. 1997). L'auteur a identifié un ensemble d'aspects non fonctionnels. Ces aspects sont: les types de données, la conception détaillée des PBOs et la configuration du système de PBO. Bien que la séparation (visuelle) des préoccupations de point de vue conception est très bénéfique, elle ne rejoint pas la définition des aspects tel qu'ils ont été définis par la communauté de la POA. Dans (Carley & Stewart 2001), le tisseur d'aspect semble jouer le rôle d'un générateur de code (compilateur + éditeur de liens) derrière une interface utilisateur graphique

¹Executable and Linkable Format

(GUI), alors que l'utilisateur (le développeur) n'a aucun contrôle sur les points de jonction (*joinpoints*) et les greffons (*advices*).

VEST (Stankovic et al. 2003) est un outil permettant de vérifier et d'analyser la dépendance entre composants lors du développement de systèmes temps réel distribués conçus initialement à base de composants. L'outil fournit un environnement graphique dans lequel le comportement temporel des modules d'un système temps réel peut être spécifié et analysé (par exemple les dates-limites et les périodes de l'algorithme WCET). Un avantage très important dans VEST c'est que les aspects sont indépendants du langage de programmation. Ils sont écrits (spécifiés) à l'aide d'un langage de script appelé VPAL (Stankovic et al. 2003).

VEST est orienté vers le développement de systèmes temps réel spécifique (avionique) . Les composants utilisés par l'outil dans le développement ne sont pas standards (dédiés *Bold Stroke* de *Boeing*).

Pour la personnalisation des systèmes d'exploitation temps réel (RTOS), Park et al. (Park et al. 2003, Park & Hong 2003) présentent un framework orienté aspects basé sur UML. Le framework fournit un environnement graphique intégrant quatre diagrammes : un diagramme d'aspect (*aspect diagram*), un diagramme de classes entrecroisées (*crosscutting class diagram*), un diagramme de structure de classe (*class structure diagram*) et un diagramme de structure de méthode (*method structure diagram*). Les aspects ont été conçus dans le framework d'une manière hiérarchique. L'idée de ce framework est basée sur le concept de l'*open class* (Clifton et al. 2000, Millstein & Chambers 2002). Une *open class* est une classe à laquelle une méthode peut être ajoutée sans éditer la classe directement. Une méthode quand à elle est composée de blocs de base (*basic blocs*). Chaque bloc de base appartient à un ou plusieurs aspects. Chaque aspect représente une caractéristique qui peut être activée ou désactivée dans le framework. Un aspect est caractérisé par des méthodes et des attributs qui peuvent être encapsulés dans une ou plusieurs classes.

Le framework présenté par Park est destiné au développement de systèmes d'exploitation temps réel mais il n'est pas limité à ces derniers, le framework peut être utilisé pour le développement d'autres systèmes. D'autre part, le framework ne présente aucune modélisation de l'aspect temporel.

Dans (Yokoyama 2005) l'auteur présente une méthode de développement orienté aspect pour le développement de systèmes de contrôle embarqués. La méthode est basée sur le traitement piloté par le temps (time-triggered processing) et le traitement piloté par les événements (event-triggered processing). Le processus de conception consiste à séparer

la conception fonctionnelle de la modélisation dynamique du système. Les aspects définis dans le model dynamique sont par la suite tissés dans les classes définies dans le model fonctionnel.

En séparant le model dynamique du modèle statique dans la conception du système, cette méthode isole l'aspect temporel du modèle fonctionnel mais il reste toujours lié au modèle comportemental.

Dans (Zhang & Liu 2005) l'auteur propose une méthode orientée aspect, de modélisation des systèmes temps réel, basée sur UML. Selon l'auteur un système temps réel peut être vu comme une structure statique avec un comportement dynamique et un aspect temporel. L'auteur propose de séparer l'aspect temporel dans la modélisation des systèmes temps réel de la structure statique et du comportement dynamique. L'auteur propose un modèle temporel qui consiste à étendre UML avec des stéréotypes. Pour le tissage de l'aspect temporel l'auteur utilise un *statechart* concurrent au *statechart* spécifiant le modèle dynamique.

L'utilisation du *statechart* pour le tissage de l'aspect temporel permet de s'emparer du développement d'un « vrai » tisseur d'aspect, par contre ceci ne permet pas une séparation « complète » de l'aspect temporel des autres fonctionnalités du système.

3 APPROCHE POUR LA MODÉLISATION DES SYSTÈMES TEMPS RÉEL

La mise en œuvre des systèmes temps réels fait intervenir deux aspects : un aspect fonctionnel et un aspect non-fonctionnel. Le premier est lié au service rendu par l'application, par exemple les mécanismes de contrôle/commande des actionneurs d'une aile d'avion, les mécanismes d'assistance au freinage, etc. En revanche, le second aspect concerne les mécanismes propres au temps réel, à savoir la synchronisation des horloges, l'ordonnancement des tâches, le respect des échéances, etc. L'intégration de ces deux aspects rend la mise en œuvre d'un tel système délicate et sa maintenance difficile.

Pour pallier ce problème, nous proposons l'application du principe de séparation des préoccupations (*Separation of Concerns* ou *SoC*) dans la phase de modélisation des systèmes temps réel (Machta et al. 2009). Ceci conduit à modéliser séparément les aspects fonctionnels et les aspects non-fonctionnels afin de les intégrer par la suite. Nous utiliserons la technique orientée aspect pour « tisser » les besoins non fonctionnelles sur le modèle des besoins fonctionnels que nous appelons, le long de cet article, application standard. Cette proposition est le cœur de notre approche. Elle est illustrée par la figure 1.

Conformément à notre approche, les applications

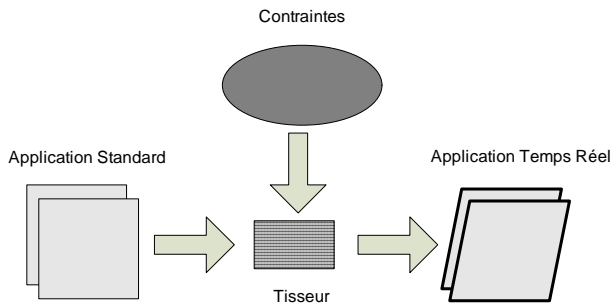


Figure 1: Tissage des contraintes sur une application standard

temps réel peuvent être développées à partir de zéro. Dans ce cas, les besoins fonctionnels et non-fonctionnels sont développés séparément. Le modèle non-fonctionnel sera tissé sur le modèle fonctionnel par le tisseur. Les applications temps réel peuvent aussi être développées à partir de certaines applications standards. Dans ce cas, il faudra définir et concevoir les besoins non-fonctionnels à tissés sur l'application standard.

Le processus de tissage se fait à la phase de conception. Ce choix offre une approche flexible et indépendante du langage de programmation. En effet, lorsque le tissage se fait à la phase de conception, aucune contrainte de langage de programmation ne se présente, ce qui donne plus de liberté dans le choix du langage d'implémentation. De plus, l'introduction des contraintes temporelles à la phase de conception peut changer la topologie de l'application alors que la réorganisation de l'application n'est pas possible à la phase de compilation. Par exemple, lorsque des tâches ne satisfont pas leurs contraintes temporelles après simulation ou exécution du modèle de l'application nous pouvons intervenir pour changer le couplage des tâches avec des objets.

Dans notre approche de Modélisation Orientée Aspect (MOA), les points de jonction (joinpoint), les greffons (advice) et les aspects ont le même rôle que dans la programmation par aspects, mais ils sont exprimés différemment.

En MOA, un point de jonction fait référence à un élément ou un ensemble d'éléments (nœud ou ensemble de nœuds) du modèle. Un greffon peut être un élément ou une propriété d'un élément, selon le point de jonction.

Le processus de tissage est basé sur un ensemble de règles. Ce dernier spécifie comment la structure ou le comportement de la préoccupation sera adapté. L'adaptation peut être statique ou dynamique. Dans le premier cas, la règle agit sur la description statique de la préoccupation. Dans le deuxième, la règle adapte la description dynamique du système. Nous

introduisons dans ce papier une premier patron de règle. Ce patron est instancié afin de définir les greffons à tisser tissés sur le modèle (1):

$$R_1 = JoinPoint.WeaverAction(Constraint). \quad (1)$$

Dans R_1 , *joinpoint* indique l'endroit où les contraintes seront tissées. *WeaverAction* spécifie l'action exécutée par le tisseur. *Constraints* définit le type de la contrainte à tisser.

4 CONTRAINTES TEMPS RÉEL MARTE ET APPLICATION UML

4.1 Définition du processus

Dans cette section nous proposons d'appliquer notre approche à la modélisation orientée objet. Dans ces dernières années, UML a connu un grand succès et s'est imposé comme le langage de modélisation le plus utilisé. Il a été appliqué dans des différents domaines et, en particulier, la modélisation des systèmes temps réel et embarqués. Mais certaines propriétés de ces systèmes n'ont pas pu être bien exprimées avec UML, comme les contraintes temporelles par exemple. Les concepteurs de ces systèmes utilisaient leurs propres extensions à UML (i.e. profiles) pour spécifier leurs applications, ce qui a fait éloigner UML de son avantage principal ; langage de modélisation unifié. Récemment, le groupe OMG a standardisé un profil d'UML pour la modélisation et l'analyse des systèmes temps réel et embarqués (i.e. MARTE, Modeling and Analysis of Real-Time and Embedded systems) (OMG 2007). Le profil MARTE est une extension d'UML permettant la modélisation des concepts temporels, des ressources génériques et des mécanismes d'allocation. L'application de notre approche consiste à tisser des contraintes temps réel sur une application standard en UML pour produire en sortie une application temps réel en MARTE. La Figure 2 montre comment une application temps réel peut être générée par tissage des contraintes temps réel MARTE sur une application standard modélisé avec UML.

Comme illustré dans la figure, les contraintes temps réel (i.e. CTR MARTE) et l'application standard (i.e. STD UML) sont exportés au format XMI (i.e. XML Metadata Interchange). XMI² est un format standard et interopérable permettant d'exporter (respectivement importer) des modèles UML à partir (respectivement vers) un document XML. Cette transformation ne conserve les spécifications des préoccupations.

Les contraintes orientés aspect (i.e. contraintes OA)

²Tous document XMI est un document XML. Nous utilisons le terme XML à l'instar de XMI le long de ce papier.

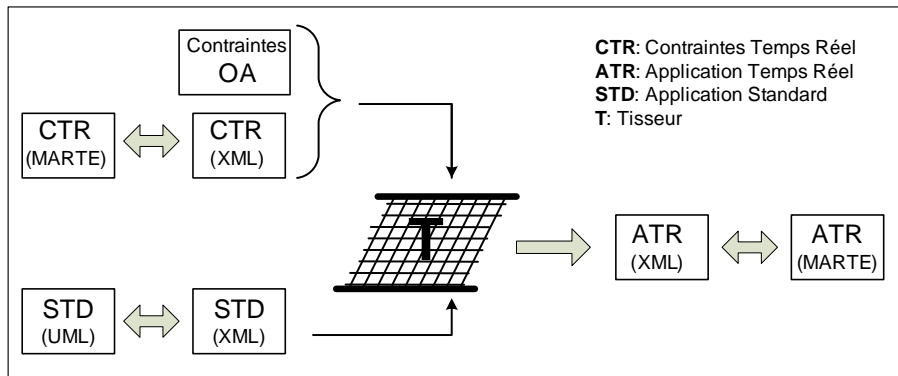


Figure 2: Phases du processus de tissage

présentent une description des points de jonction, griffons, adaptation ... etc (Schauerhuber et al. 2007) qui ne sont pas familiers au concepteur temps réel. Pour cacher ces mécanismes au concepteur temps réel, les contraintes orientées sont couplés au document XML contenant les contraintes temps réel. Les deux documents XML sont tissés pour générer le document XML de l'application temps réel qui sera importé, par la suite, à un modèle MARTE.

Le processus de tissage est basé sur l'utilisation d'un ensemble d'instances de la règle R_1 appliquées au modèle UML. Comme cité plus haut, *JoinPoint* définit un élément du modèle, une classe par exemple. *WeaverAction* par contre, est une action exécutée sur le *JoinPoint* (une classe dans ce cas) comme l'ajout d'un attribut par exemple. Finalement *Constraint* peut être la valeur par défaut ou le type de l'attribut. Des exemples d'instances de R_1 appliquées au modèle UML sont présentées dans notre étude de cas.

4.2 Etude de cas

4.2.1 Présentation de l'application

Pour illustrer l'application de notre approche nous retenons l'exemple de la commande d'un robot télécommandé qui est largement utilisé dans la littérature des systèmes embarqués et temps réel. L'exemple est utilisé pour montrer comment tisser des contraintes temps réel sur le modèle UML. Le robot est utilisé lorsque la présence de l'homme n'est pas nécessaire. Il recueille des informations en utilisant ses capteurs et les transmet au contrôleur (*Controller*). Le contrôleur communique des informations au poste de commande (*Command Station*) qui les interprète et affiche les données sur l'état du robot. Le robot est connecté au contrôleur au moyen du bus VME, par contre, le contrôleur et le poste de commande communiquent via un bus CAN.

Nous nous intéressons à la conception logicielle de la station de commande. Elle se compose d'interfaces graphiques permettant à l'opérateur de commander le robot et d'afficher des informations recueillies par le robot. La figure 3 illustre une architecture logicielle

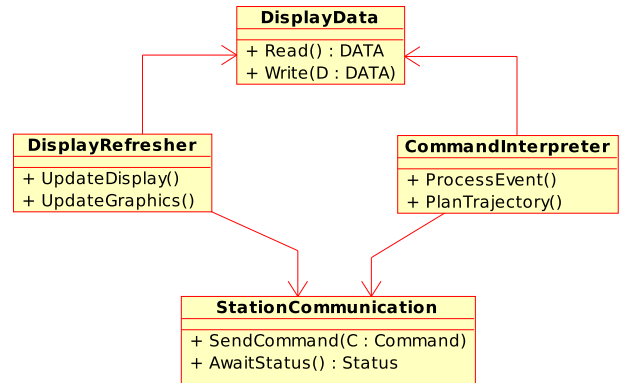


Figure 3: Modèle UML de la station de commande

possible pour la station de commande.

La classe *DisplayRefresher*, met à jour d'interface graphique de données en interprétant les messages d'état qu'elle reçoit via le bus CAN. La classe *StationCommunication* gère la communication via le bus CAN. La classe *DisplayData* fournit les données partagées à d'autres classes. La classe *CommandInterpreter* poignées exploitant événements générés par les éléments de contrôle de l'interface graphique. La classe *CommandInterpreter* exploite les événements générés par les éléments de contrôle d'interface graphique.

4.2.2 Tissage sur diagramme de structure: diagramme de classe

Dans un environnement temps réel, chaque objet instancié des classes *CommandInterpreter* et *DisplayRefresher* peut être couplé à une tâche distincte. Ils peuvent (ces objets) aussi introduire des données simultanément à la classe *DisplayData*. Alors, la classe *DisplayData* sera considérée comme section critique. La figure 4 illustre le nouveau modèle, dans un contexte temps réel, avec MARTE.

Dans le modèle MARTE (i.e. Figure 4) les deux classes *CommandInterpreter* et *DisplayRefresher* ont été décorés avec le stéréotype *SchedulableResource*. *DisplayData* était stéréotypé avec *MutualExclusionResource*. En MARTE, l'instance de type

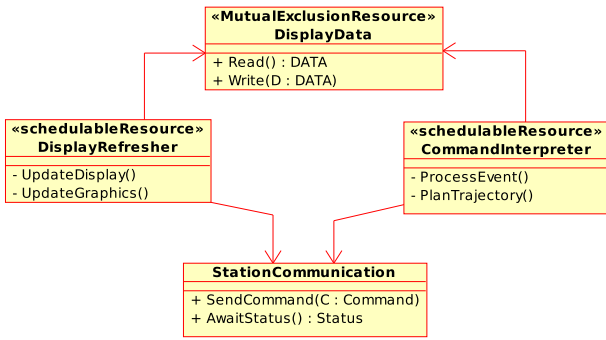


Figure 4: Modèle UML MARTE de la station de commande

SchedulableResource est associée à un ordonnanceur et possède le pouvoir de déclencher une action de contrôle. *MutualExclusionResource*, cependant, est une ressource protégée associée à des paramètres de protection et d'un protocole associé à l'ordonnanceur qu'il gère.

Pour cet exemple (i.e. poste de commande de robot télé-opéré), l'adaptation des préoccupations consiste en des décorations de classes avec des stéréotypes appropriés, spécifiés par un concepteur de système temps réel.

Les contraintes de tissage et l'adaptation du concepteur temps réel sont ensuite associées par le tisseur en utilisant des instances de règle R_1 . La figure 5 présente les instances appliquées à notre exemple.

Dans notre exemple, les contraintes concernent le type de classe qui est spécifié par les stéréotypes dans le langage UML. Cette adaptation est structurelle. Le *JoinPoint* dans ce cas est le nom de classe (en l'occurrence *DisplayRefresher*, *CommandInterpreter* et *DisplayData*). *WeaverAction* cependant, est le stéréotypage des classes. Enfin, *Constraint* est le nom de stéréotype MARTE qui est *SchedulableResource* pour les deux premières classes et *MutualExclusionResource* pour la classe *DisplayData*.

La Figure 6 présente la structure générale³ du document XML du modèle MARTE. On peut voir, dans la figure 6, les changements effectués par le tisseur au niveau des lignes [7,11] et [38,45]. Ces lignes contiennent des données nécessaires pour l'expression et la mise en œuvre des éléments MARTE. Dans la même figure on peut voir l'application des contraintes définies par les instances de la règle (5)

³Pour des raisons de lisibilité certains paramètres et valeurs ont été modifiés

```

DisplayData . Stereotyped ( MutualExclusionResource )
CommandInterpreter . Stereotyped ( SchedulableResource )
DisplayRefresher . Stereotyped ( SchedulableResource )
  
```

Figure 5: Instances de la règle R_1

au niveau des lignes [47,49]. Par exemple, dans la balise `<GRM:MutualExclusionResource>` de la ligne 47, l'attribut `base_Classifier` prend comme valeur `DDid` qui est l'identité de la classe *DisplayData* dans le document XML.

Notre approche est basée sur la forme XML d'un modèle UML. Il est facile donc, d'appliquer le processus de tissage à d'autres diagrammes UML. Seules les instances de la règle changent d'un diagramme UML à un autre, et le principe de tissage reste le même.

4.2.3 Tissage sur diagramme de comportement: Diagramme de séquence

La figure 7 présente un exemple de scénario de l'exécution possible du modèle de la station de commande. L'opérateur lance une nouvelle commande à travers l'interface graphique. Ce ci déclenche la méthode *ProcessEvent* de l'objet *CommandInterpreter*.

Ce dernier, lit les données à partir de l'objet *DisplayData*, planifie la trajectoire du robot et envoie une commande par l'intermédiaire de l'objet *StationCommunication*.

Le profil MARTE permet d'ajouter une observation du temps (*TimeObservation*) dans le diagramme de séquence du modèle. Une observation du temps est une référence temporelle à un instant donné durant l'exécution. Elle est utilisée pour exprimer des contraintes temporelles dans le modèle MARTE.

La figure 8 présente un exemple d'utilisation d'une "observation du temps" pour exprimer une contrainte temporelle. Une observation de temps appelée *Init* est associée à l'invocation de la méthode *ProcessEvent*. La deuxième observation du temps appelée *SendCom* est associée à l'invocation de la méthode *SendCommand*. *Init* et *SendCom* sont utilisées comme des variables pour définir une contrainte temporelle.

Les instances de règle permettant le passage du diagramme de séquence de la figure 7 à celui de la figure 8 sont illustrés par la figure 9.

5 CONCLUSION

Dans ce papier nous avons présenté une approche de conception d'applications temps réel et embarqués. Elle est basée sur la séparation des aspects fonctionnels de ceux non-fonctionnels à la phase de conception. Nous avons utilisé les concepts de modélisation orientée aspect pour regrouper les aspects fonctionnels et aspects non-fonctionnels. L'application de cette technique permet d'augmenter la productivité et de réduire les coûts de développement des applications temps réel.

Nous avons illustré l'application de notre approche au domaine de la modélisation orienté objet et en

```

2 <?xml version="1.0" encoding="UTF-8"?>
  <!--XMI Header-->
  <xmi:XMI xmi:version="2.1" xmlns:xmi="..." xmlns:xsi="..." >
    <!--UML MODEL-->
    <uml:Model xmi:id="xmiMODELid" name="ModelName">
7      <!--UML MARTE Resources-->
      <packageImport xmi:id="..." >
        <importedPackage href="pathmap:.../" >
        </packageImport>
      ...
12    <!--DisplayData class definition-->
    <packagedElement xmi:type="uml:Class" xmi:id="DDid" name="DisplayData">
      <ownedOperation xmi:id="..." name="Read">
      ...
      </ownedOperation>
17    ...
    </packagedElement>
    <!--CommandInterpreter class definition-->
    <packagedElement xmi:type="uml:Class" xmi:id="CIid" name="CommandInterpreter">
      <ownedOperation xmi:id="..." name="ProcessEvent" />
22      <ownedOperation xmi:id="..." name="PlanTrajectory" />
      ...
    </packagedElement>
    <!--DisplayRefresher class definition-->
    <packagedElement xmi:type="uml:Class" xmi:id="DRid" name="DisplayRefresher">
27      <ownedOperation xmi:id="..." name="UpdateDisplay"/>
      <ownedOperation xmi:id="..." name="UpdateGraphics"/>
      ...
    </packagedElement>
    <!--StationCommunication class definition-->
32    <packagedElement xmi:type="uml:Class" xmi:id="Scid" name="StationCommunication">
      <ownedOperation xmi:id="..." name="SendCommand">
      ...
      </ownedOperation>
      <ownedOperation xmi:id="..." name="AwaitStatus"/>
37    </packagedElement>
    <!--UML MARTE Applied profiles-->
    <profileApplication xmi:id="...">
      <eAnnotations xmi:id="..." source="...">
        <references xmi:type="ecore:EPackage" href="pathmap:..."/>
42      </eAnnotations>
      <appliedProfile href="pathmap:..."/>
    </profileApplication>
    ...
47 </uml:Model>
    <GRM:MutualExclusionResource xmi:id="..." base_Classifier="DDid"/>
    <GRM:SchedulableResource xmi:id="..." base_Classifier="CIid"/>
    <GRM:SchedulableResource xmi:id="..." base_Classifier="DRid"/>
  </xmi:XMI>

```

Figure 6: Une partie du document XML correspondant au modèle UML MARTE

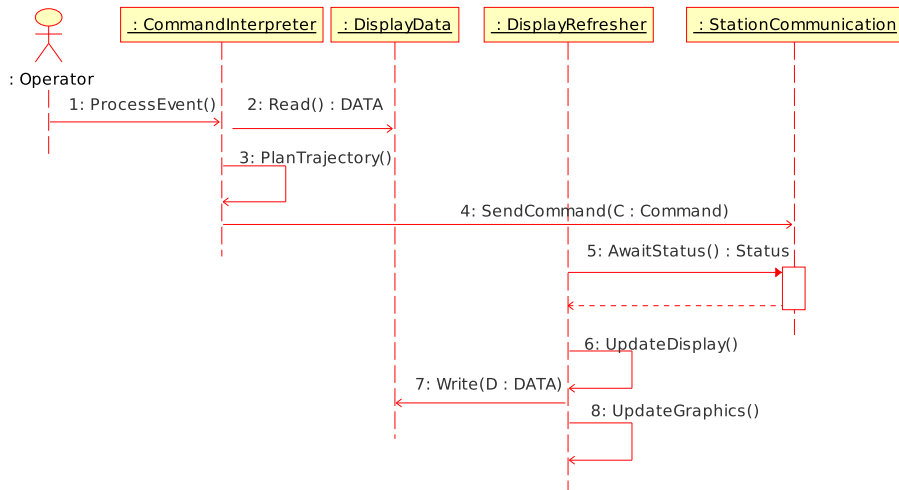


Figure 7: Diagramme de séquence illustrant un exemple de scénario d'exécution

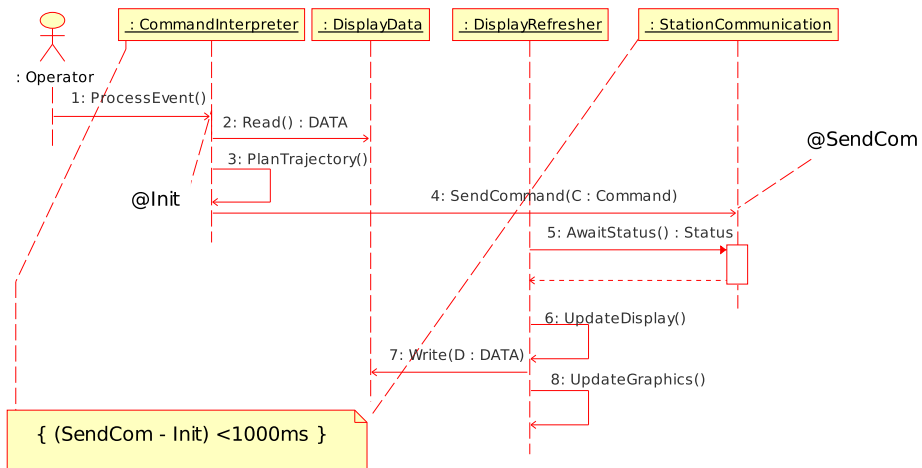


Figure 8: Diagramme de séquence de la station de commande en MARTE

```

Operator : ProcessEvent >: CommandInterpreter . addTimeObservation ( Init )
CommandInterpreter : SendCommand >: StationCommunication . addTimeObservation ( SendCom )
CommandInterpreter : [] : StationCommunication . addConstraintNote ( ( SendCom - Init ) <100ms )
    
```

Figure 9: Instances de règle pour tissage de contraintes sur diagramme de séquence

particulier au cas du langage de modélisation UML. Nous avons traité un cas d'étude auquel nous avons appliqué le tissage des contraintes temps réel sur deux diagrammes différents.

Le tissage des contraintes temps réel sur différents diagrammes et vues du modèle peut engendrer des conflits dans la génération des instances de la règle présentée. Nous allons travailler sur les méthodes et les techniques de résolution des conflits. Nous envisageons aussi l'intégration de l'outil développé dans un IDE supportant UML et UML MARTE (Papyrus (Papyrus n.d.) par exemple).

References

- Carley, T. & Stewart, D. (2001). Visual aspect-oriented programming of resource constrained real-time embedded systems using the port-based object model of computation, *OOPSLA Workshop on Domain Specific Visual Language, October*.
- Clifton, C., Leavens, G., Chambers, C. & Millstein, T. (2000). MultiJava: modular open classes and symmetric multiple dispatch for Java, *Proceedings of the 15th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications* pp. 130–145.
- Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C. V., Loingtier, J. & Irwin, J. (1997). Aspect-oriented programming, *In Proceedings of European Conference on Object-Oriented Programming*.
- Machta, N., Bennani, M. & Ben Ahmed, S. (2009). Aspect oriented design of real-time applications, *Industrial Informatics, 2009. INDIN 2009. 7th IEEE International Conference on*, pp. 763–767.
- Millstein, T. & Chambers, C. (2002). Modular Statically Typed Multimethods, *Information and Computation* **175**(1): 76–118.
- OMG (2007). Specification. A UML Profile for MARTE, Beta 1.
- Papyrus (n.d.). <http://www.papyrusuml.org>, Access 28-9-2009.
- Park, J. & Hong, S. (2003). Customizing Real-Time Operating Systems with Aspect-Oriented Programming Framework, *SOC Design Conference* **970**.
- Park, J., Kim, S. & Hong, S. (2003). Weaving aspects into real-time operating system design using object-oriented model transformation, *Object-Oriented Real-Time Dependable Systems, 2003. Proceedings. Ninth IEEE International Workshop on* pp. 292–298.
- Schauerhuber, A., Schwinger, W., Retschitzegger, W., Wimmer, M. & Kappel, G. (2007). A survey on aspect-oriented modeling approaches, *Technical report*, Faculty of Informatics, Vienna.
- Stankovic, J., Zhu, R., Poornalingam, R., Lu, C., Yu, Z., Humphrey, M. & Ellis, B. (2003). Vest: An aspect-based composition tool for real-time systems, *RTAS '03: Proceedings of the The 9th IEEE Real-Time and Embedded Technology and Applications Symposium*, IEEE Computer Society, Washington, DC, USA, p. 58.
- Stewart, D., Volpe, R. & Khosla, P. (1997). Design of dynamically reconfigurable real-time software using port-based objects, *Software Engineering, IEEE Transactions on* **23**(12): 759–776.
- Yokoyama, T. (2005). An aspect-oriented development method for embedded control systems with time-triggered and event-triggered processing, *RTAS '05: Proceedings of the 11th IEEE Real Time on Embedded Technology and Applications Symposium*, IEEE Computer Society, Washington, DC, USA, pp. 302–311.
- Zhang, L. & Liu, R. (2005). Aspect-oriented real-time system modeling method based on UML, *Embedded and Real-Time Computing Systems and Applications, 2005. Proceedings. 11th IEEE International Conference on* pp. 373–376.