Rapport intérmediaire de stage: Le langage synchrone LS

Domnica Pavel

Encadreur:Annie Ressouche
Projet Pulsar, INRIA Sophia Antipolis

Juin 2008

1 La platforme WComp

WComp est une platforme developée par les étudiants de l'Université de Nice - Sophia Antipolis. Cette platforme permet de gérer les communications et les informations entre les dispositifs UPnP (sorte de Web Services pour dispositifs) sur la base d'un assemblage de composants.

Un composant est une instance de classe (Java ou C selon l'implémentation). Il est doté d'entrées et de sorties ; typiquement des interfaces serveur qui sont des ensembles d'opérations et des interfaces client qui se composent d'évènements qui peuvent être émis.[Daniel Cheung Foo Wo ...]

Les composants Wcomp se découpent en deux catégories : les composants logiciels (appelés simplement composants) et les composants mixtes c'est à dire des composants logiciels représentant un équipement matériel.

La communication entre les composants est de type asynchrone. Les composants émettent des événements via les interfaces client. D'autres composants sont mis en écoute de ces événements. Ils sont notifiés via leur interface serveur par une transformation adéquate de l'événement en un envoi de message. La transformation est prise en compte par le composant émetteur. Dans un assemblage, on différencie les composants dits actifs des composants passifs. Les composants Wcomp sont qualifiés de passifs, s'ils ne créent pas de nouveaux fils d'exécution (thread) ou de tâches, et actifs dans le cas contraire. On peut noter qu'il est possible de rendre actif un composant passif (en l'associant avec un composant actif) ou de passifier un composant actif (grâce à un composant Wcomp de bufferisation).[Daniel Cheung Foo Wo ...]

Wcomp fonctionne sur un découpage Designer/Container. Le Container a en charge la gestion des composants qui forment l'application finale. C'est lui qui met en place les connections entre les différents briques applicatives. Le Designer est une application qui communique avec

le Container afin de le diriger. C'est par le Designer que l'on ordonne au Container d'ajouter/retirer des composants. C'est aussi le Designer qui indique les liaisons à créer/détruire.

2 Langages Synchrones

La partie théorique sur Langages Synchrones est rédigée à partir des documents de Charles André. [...]

Dans les années 1980, des études indépendantes ont conduit à l'apparition de langages spécialisés dits langages synchrones comme Esterel, Lustre et Signal, ainsi qu'à celle d'un formalisme graphique appelé Statecharts. Ces langages et formalismes répondaient à des besoins de programmation et de modélisation des systémes réactifs, c'est à dire des systèmes qui sont en interaction permanente avec leur environnement. La nature des applications considérées explique que ces langages aient été initialement con cus par des équipes mixtes comprenant des automaticiens et des informaticiens.

Visant des applications industrielles, il est vite apparu que ces langages devaient être expressifs et compilables de fa con efficace afin de pouvoir traiter de "gros" programmes. Les techniques de compilation des langages synchrones ont fait de grands progrès dans les années 1990, permettant ainsi à ces langages de passer du statut académique au statut industriel. Leur champ d'application s'est également élargi. Ils sont à présent largement utilisés dans le développement de logiciels embarqués ainsi que dans la conception de circuits numériques complexes.

Les langages synchrones sont spécialisés dans l'expression rigoureuse des réactions. Un programme synchrone doit apporter des réponses non ambigu es aux questions : quelles sont les actions qui composent une réaction, quand les exécuter, quelles sont les liens de dépendance entre ces actions, ... Une caractéristique essentielle des langages synchrones est de pouvoir spécifier des comportements déterministes même en présence de parallélisme et de communications. La chaîne de transformation depuis la spécification jusqu'à l'implémentation préserve ce déterminisme.

En fonction de la classe d'application visée, l'utilisateur peut choisir son style de programmation : un style impératif (pour les applications dans lesquelles les aspects relatifs au contrôle sont prépondérants) ou un style déclaratif (pour des applications avec une part importante de traitements de données en temps réel). Cette distinction a tendance à s'estomper avec les nouvelles plates-formes de développement synchrones qui permettent de mélanger les deux styles de programmation. La plupart des langages offrent une syntaxe textuelle et une syntaxe graphique. Dans tous les cas, les langages restent simples avec un nombre limité d'entités et de constructeurs. Malgré leur simplicité, ces langages permettent l'utilisation de concepts de haut-niveau comme le parallélisme de conception, les communications par diffusion, et les descriptions hiérarchiques.

3 Le langage LS

Par le langage LS on veut modéliser les connexions qui relie d'appels de méthodes à un événement.

On décrit une sémantique opérationnelle du langage synchrone LS dans lequelle on modélise les connexions qui connectent d'appels de méthodes à un évènement.

On veut respecter l'hypothèse synchrone : une réaction est atomique. Pendant une réaction, la valeur des événements d'entré ne peut pas changer. Tous les évènementes sont simultanés. Ainsi, c'est pourquoi nous pouvons appeler une réaction, un instant. La succession des instants détermine une temps logique.

Une réaction comprend une phase de lecture des informations d'entrée, une phase de calcul et une phase d'écriture des informations de sortie.
Durant la réaction, le programme est complètement isolé de son environnement.

Nous avons d'abord décrit la syntaxe (l'ensemble de la grammaire du langage, qui est détaillée en Annexe I).

Les unités de programmation en langage LS on a appelée connexion (comme j'écrit la grammaire en anglais, on a connection interface et connection body) :

CONNECTION OF composant name :
connection interface
connection body
END

L'interface décrit les objects qui permettent à la connexion de comuniquer avec l'environnement : l'ensemble des environnements d'entrée (les méthodes) et l'ensemble des environnements de sortie (les événements) et le corps est exprimée utilisant des opérateurs.

3.1 Les objets du langage

La déclaration des objets se fait dans l'interface.
L'ensemble des environnements d'entrée décrit l'ensemble des méthodes d'entrée qui sont présentes, indique donc le statut de toute entrée. LS est un langage synchrone réagissant à une méthode d'entrée en produisant dans le même instant un événement de sortie.

3.2 Les instructions

L'exécution d'une instruction est caractérisée par son instant de démarrage et son instant de terminaison. Si les deux co incident, l'exécution termine instantanément sinon elle s'étend sur plusieurs réactions.

Les instructions qui composent le corps décrivent classiquement son comportement. Ces instructions du langage LS sont les opérateurs : nothing, parallel, sequence et connection. En outre, les opérateur nous aident a construire des instructions composites.

Nothing : Cet instruction se termine instantanément.

Parallèle : La parallèle $P1 \| P2$ commence l'exécution de ses deux sous-instructions dans le même instant et termine quand, à la fois P1 et P2 ont terminé. Lorsque les deux sous - instructions sont instantanées, la parallèle est instantanée.

L'exécution d'une instruction parallèle est terminée lorsque toutes les branches sont terminées.

Séquence : Dans la séquence d'instruction $P1 \gg P2$, au premiere fois est exécute sous-instruction P1. Si P1 est fini instantanément la séquence exécute immédiatement le sous-instruction P2 et s'arrête quand le sous-instruction P2 s'arrête. Si P1 s'arrête, la séquence s'arréte. La séquence se termine lors du même instant que son deuxiéme sous-instruction P2 se termine. Si les deux sous-instructions sont fini instantanées, la séquence se termine instantanément.

Connexion : $e_c \Rightarrow C_i.m_i$ c'est une connexion qui relie d'appels de méthodes à un évènement.

4 La sémantique du LS

L'hypothése synchrone s'applique aux systéme évoluant instant par instant, et rien ne peut arriver entre deux instants.

La sémantique d'un langage de programmation s'attache à donner un sens mathématique aux programmes. Il existe différentes méthodes formelles de définition de sémantique comme les méthodes opérationnelle et dénotationnelle.

Une sémantique dénotationnelle attribue un sens aux programmes d'un langage en associant à chaque construction syntaxique du langage une valeur dans un domaine de définition.

Une sémantique opérationnelle donne un sens aux programmes en terme d'étapes de calcul (ou réécritures). Quel que soit son mode de définition, une sémantique permet d'ôter toute ambiguïté dans la définition d'un langage de programmation. Elle peut aussi fournir une base pour des techniques de manipulation formelle de programmes : preuves de propriétés de correction, analyse, transformation.

La sémantique opérationnelle mélange la syntaxe et les états dans sa définition. Elle colle de très (trop) près à la syntaxe.

4.1 La sémantique comportementale du LS

La sémantique est un modéle formel pour décrire le comportement d'un programme, elle donne la signification des éléments du langage.

Pour les langages synchrones, la sémantique est définie en termes mathématiques.

Pour étudier la sémantique comportementale du LS on doit décrire le comportement de chaque opérateur. Nous avons comme opérateurs : nothing, parallèle de méthodes, séquence de méthodes, parallèle de la connexion et séquence de la connexion.

Pour étudier cela, il faut d'abord mettre en place un cadre logique qui représente un environnement (les événements et les méthodes) et puis nous définissons LS processus de calcul afin de décrire la sémantique comportementale.

Un environnement (E) est un ensemble fini de méthodes (input) et d événements (output). Un programme réagit aux environnement d'entrée (E) en produisant environnement de sortie (E'). Les relations d'entrée/sortie jouent un rôle esentiel dans la sémantique des langages synchrones.

Afin de décrire le comportement de LS, il faut d'abord introduire une algèbre de processus associes de langage.

Ensuite nous pouvons définir la sémantiques avec un ensemble de régles de reécriture qui détermine un programme d'exécution.

La sémantique officialise une réaction d'un programme P en fonction d'un environnement d'entrée $P \xrightarrow[E]{E'} P$ a le sens habituel : E et E' sont respectivement d'entrée et de sortie des environnements, le programme P réagit a E, atteint un nouvel état représenté par le P et l'environnement de production est E' .(Avec la donnée E, le programme P produit E', et il reste à exécuter P'.)

Pour calculer une telle réaction nous comptons sur la sémantique du comportement de LS. Cette sémantiques soutient une règle de base de spécification pour décrire le comportement de chaque opérateur de LS algèbre de processus associes à LS langage.

Notons avec M les méthodes et avec Ev les événements. Soit B = un ensemble booléens .

Une fonction $[] : M \to B$, qui définit la notion de terminaison d'un appel de méthode :

$$[C_i.m_i] = \begin{cases} tt \text{ si } C_i.m_i \; termine \\ ff \text{ sinon} \end{cases}$$

$Ev = (e_i)$, ou $i = 1, ..., n$, $e_i \in E$.
On note avec e_c les événements complexe qui sont une combinations des évènements : $e_c = B(e_1, e_2, ..., e_n)$

$TERM$ est un booléen vrai lorsque P se termine.

La paralèlle du méthodes :

$$\frac{e_i \Rightarrow m_1 \xrightarrow[E]{TERM_1} P', e_i \Rightarrow m_2 \xrightarrow[E]{TERM_2} Q'}{e_i \Rightarrow m_1 \| m_2 \xrightarrow[E]{TERM_1 \cdot TERM_2} (P' \| Q')}$$

La séquence du méthodes :

$$\frac{e_i \Rightarrow m_1 \xrightarrow[E]{tt} nothing, e_i \Rightarrow m_2 \xrightarrow[E]{TERM} P'}{e_i \Rightarrow m_1 \gg m_2 \xrightarrow[E]{TERM} P'}$$

La connexion relie tous les événements de méthodes.
Si on a :

$$\frac{e_c, [C.m] = tt}{e_c \Rightarrow C.m \xrightarrow[E]{tt} nothing}$$

$$\frac{e_c, [C.m] = ff}{e_c \Rightarrow C.m \xrightarrow[E]{ff} e_c \Rightarrow C.m...}$$

On peut definir **la parallèle de la connexion :**

$$\frac{e_c \Rightarrow C_1.m_1 \xrightarrow[E]{TERM_1} P', e_c \Rightarrow C_2.m_2 \xrightarrow[E]{TERM_2} Q'}{e_c \Rightarrow C_1.m_1 \| C_2.m_2 \xrightarrow[E]{TERM_1 \cdot TERM_2} P' \| Q'}$$

et **la séquence de la connexion**

$$\frac{e_c \Rightarrow C_1.m_1 \xrightarrow[E]{tt} nothing, e_c \Rightarrow C_2.m_2 \xrightarrow[E]{TERM} P'}{e_c \Rightarrow C_1.m_1 \gg C_2.m_2 \xrightarrow[E]{TERM} P'}$$

$$\frac{e_c \Rightarrow C_1.m_1 \xrightarrow[E]{ff} P'}{e_c \Rightarrow C_1.m_1 \gg C_2.m_2 \xrightarrow[E]{ff} e_c \Rightarrow P' \gg C_2.m_2}$$