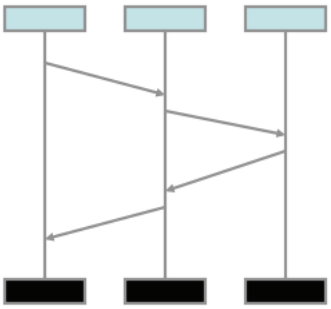
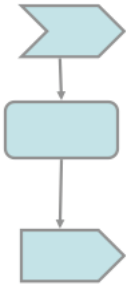
Annexe 1 : SDL et MSC





Langages de spécification

En fonction de la complexité de la spécification d'un protocole, on utilise différentes représentations graphiques pour leur description. Il existe plusieurs langages de description pouvant être utilisés. La liste suivante en présente quelques-uns :

- Data Flow Diagram (DFD)
- Message Sequence Chart (MSC)
- System Description Language (SDL)
- Unified Modelling Language (UML)
- Hardware Description Language (HDL)
- etc.

Dans le cadre de ce cours, les éléments fondamentaux sont décrits au moyen de MSC et de SDL. Dans le dernier chapitre, nous effectuerons une comparaison entre les éléments MSC et/ou SDL et les éléments UML. MSC et SDL existent aussi sous forme textuelle. Cette forme est utilisée pour échanger des MSC et/ou SDL entre différents outils de modélisation ou des machines de traduction. Dans ce cours nous ne nous concentrerons uniquement sur la forme graphique.

Message Sequence Charts

MSC est un langage de description graphique qui permet la représentation de l'échange des informations entre différents systèmes. MSC a été standardisé par l'ITU (International Telecomunication Union) et est décrit dans la recommandation Z.120. Cette recommandation contient une définition en partie informelle des MSCs. Dans le secteur de la télécommunication, les MSCs sont principalement utilisés pour la spécification des éléments suivants :

- Spécification d'un protocole
- Test d'exigence

Les MSC sont souvent utilisés en parallèle avec les diagrammes SDL. Les MSC peuvent être formalisés et leur cohérence vérifiée. Cela a un rapport avec un élément important, la

vérification formelle. Nous allons ici décrire et commenter les éléments de base des MSC. De plus, la forme formalisée sera présentée.

Un MSC simple

Dans un MSC simple, les instances des niveaux de protocole deviennent respectivement les éléments du protocole et sont représentés sous ces informations échangées. Les informations peuvent venir soit de l'instance elle-même, soit du monde extérieur (environnement). Les informations affluent de façon stricte dans la séquence montrée ci-contre par le MSC. La figure suivante illustre ceci :

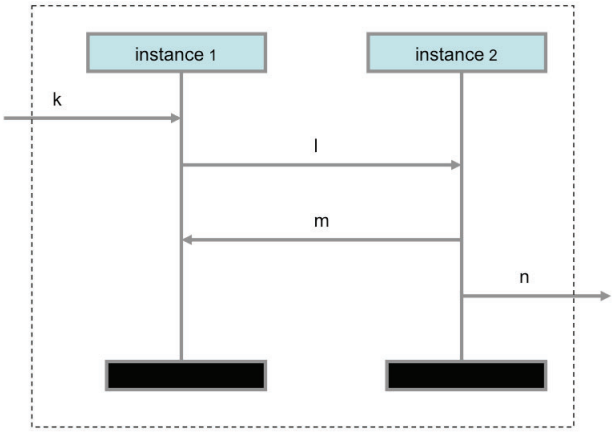


Figure 34: A simple MSC

Les lignes noires à la fin des lignes d'instance représentent aussi la fin de l'instance respective.

MSC avec des conditions

Les MSCs peuvent posséder des conditions. Cela est utile notamment dans certains scénarios qui possèdent une condition initiale et une condition finale qu'il faut préciser. Les conditions sont représentées sous la forme d'un hexagone. La figure suivante montre un MSC avec des conditions :

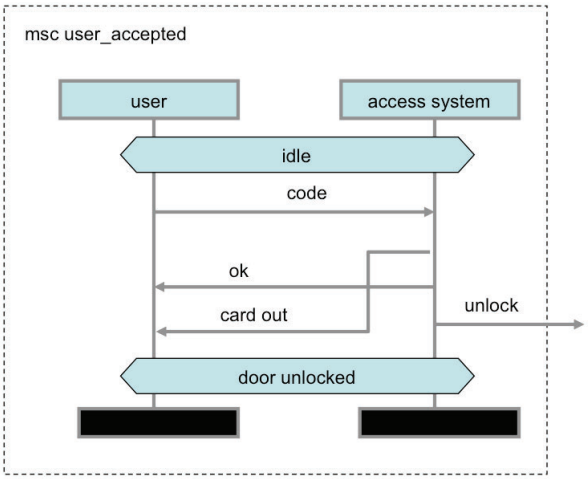


Figure 35: A MSC with conditions

MSC imbriquées

Les MSCs peuvent être organisés de façon hiérarchique. De cette manière, il est possible de diviser une MSC en plusieurs MSCs moins complexes. D'autre part, il est aussi possible de réemployer ou de référencer des parties de MSC à différents endroits. La figure suivante montre la corrélation à un MSC existant :

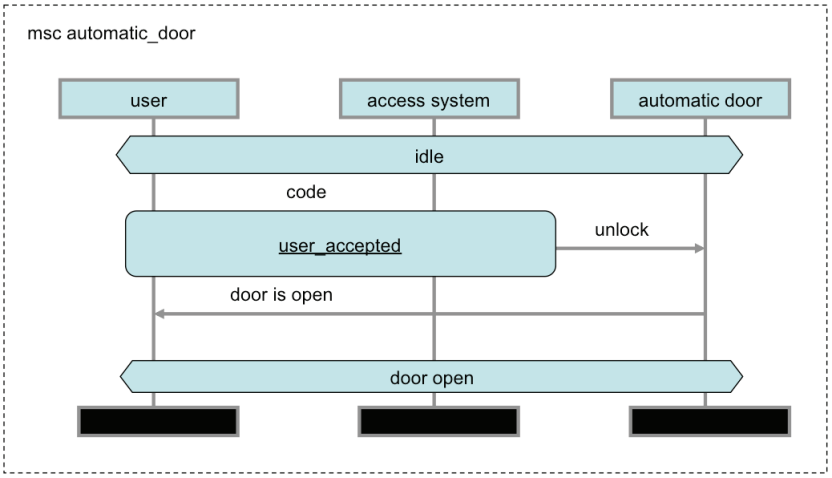


Figure 36: Layered MSC

MSCs avec des conditions temporelles

Les MSC peuvent aussi contenir des conditions temporelles. Celles-ci sont réalisées au moyen de timers et servent, avant tout, à surveiller l'arrivée d'informations ou le temps de mise en œuvre d'un MSC référencé. La figure suivante en montre un exemple :

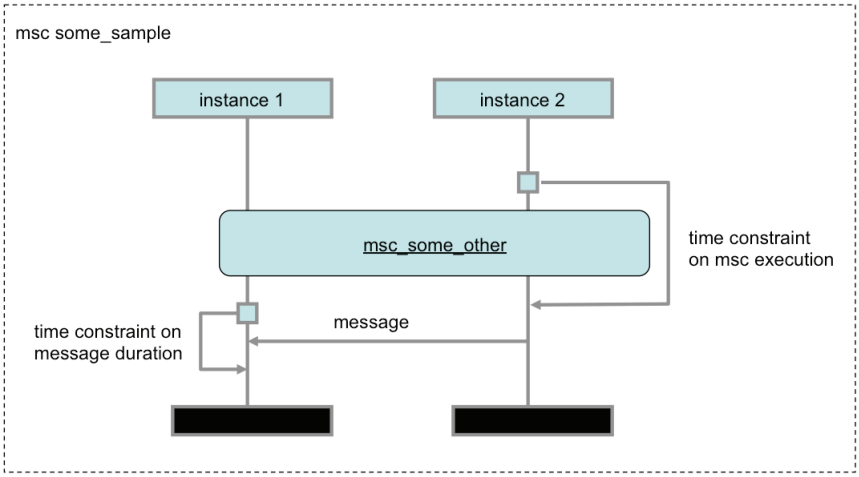
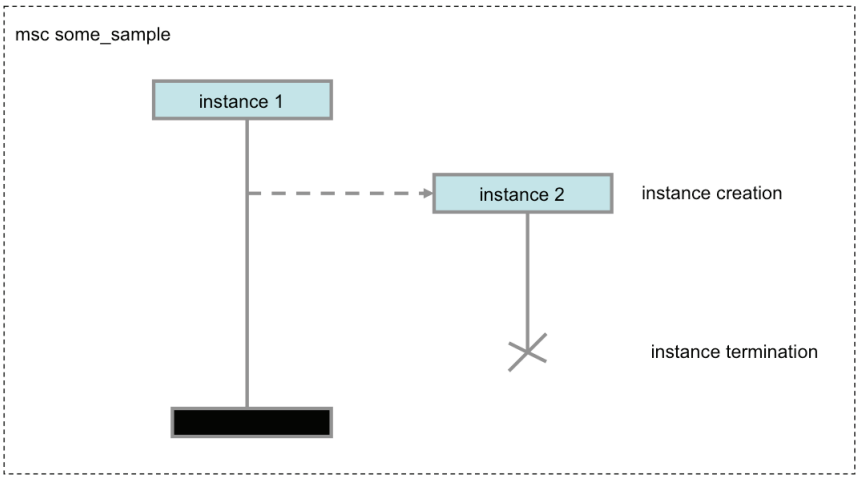


Figure 37: A MSC with time constraints

MSCs et instances

Dans un MSC, des instances peuvent être produites ou détruites. C'est typiquement le cas si l'on a besoin de créer des instances temporaires qui seront détruites par la suite. La figure suivante illustre ceci :



Appel de méthodes en MSC

Dans un MSC, il n'y a pas que des informations qui peuvent être échangées. Il est aussi possible de faire des appels de méthode. Cela a lieu lors de l'appel du mot-clé. En outre, la durée de traitement de la méthode peut être représentée par une ligne d'instance. Il est aussi possible de marquer des intervalles de temps pendant lesquels la méthode se bloque elle-même en attendant la fin d'une autre méthode. La figure suivante illustre ceci :

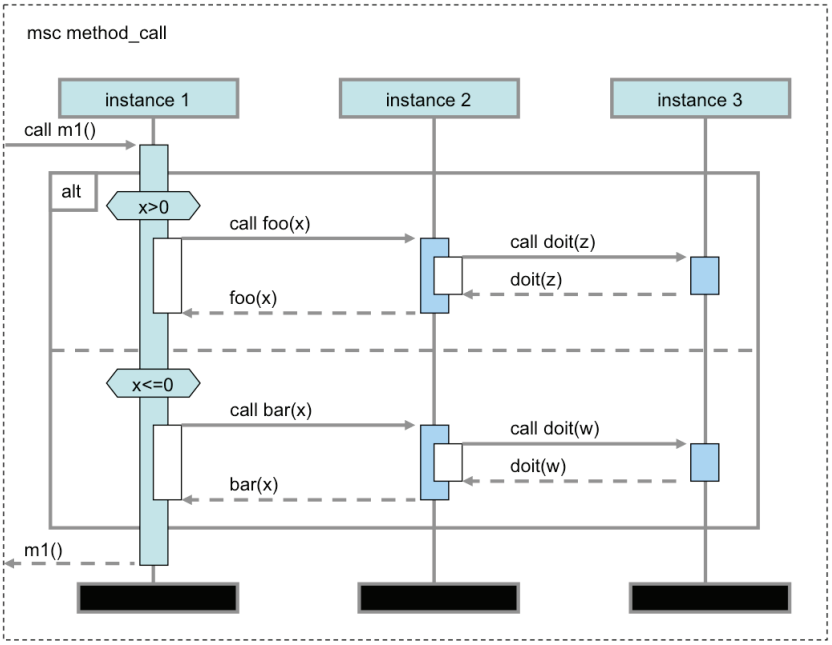


Figure 38: A MSC with method or function calls

Co-régions dans un MSC

Dans ce que l'on nomme une Co-région dans un MSC, la séquence d'information stricte est dissoute et il n'est pas possible de faire apparaître chaque information à un moment assignable à l'avance. La figure suivante en montre un exemple :

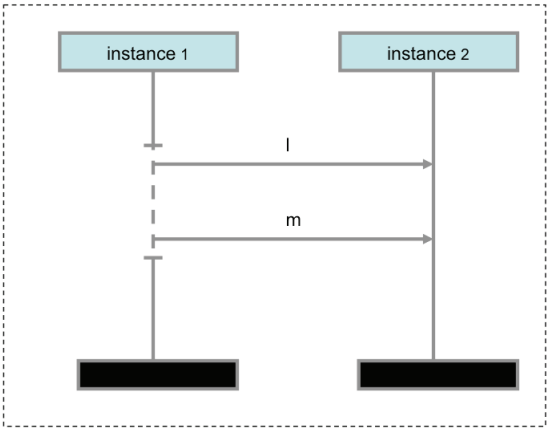


Figure 39: A MSC with co-regions

MSCs hiérarchiques (HMSC)

Les HMSC ont été introduits dans MSC96 et permettent, comme son nom l'indique, de représenter les systèmes d'une façon hiérarchique. La figure suivante en montre un exemple :

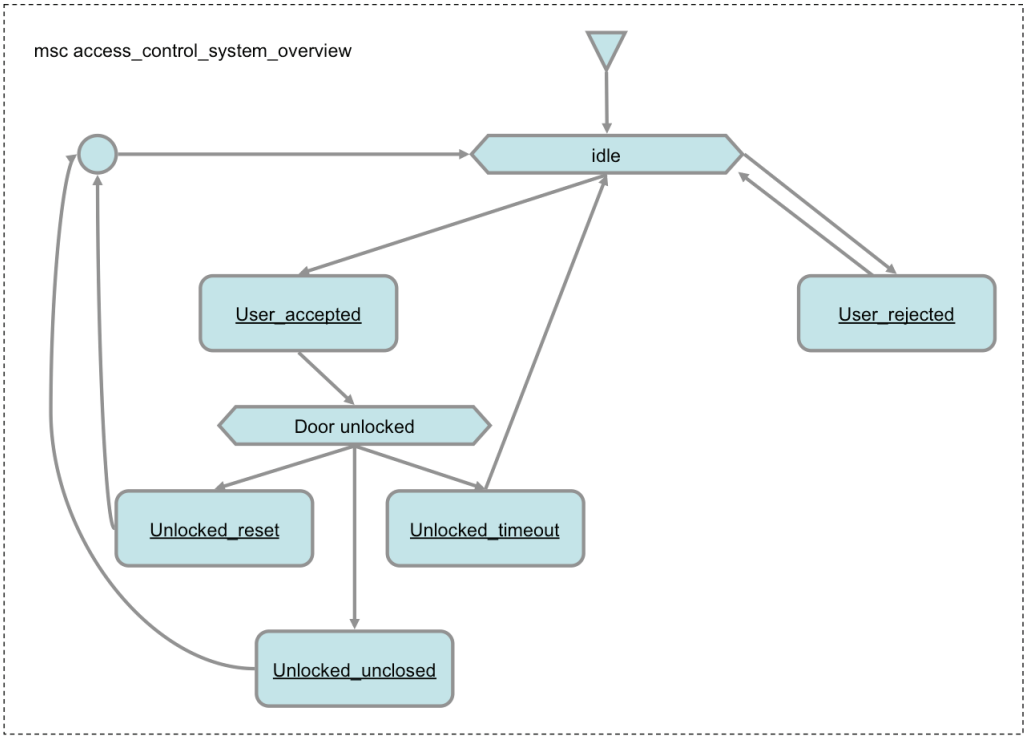


Figure 40: A hierarchical MSC (HMSC)

Pensées finales

MSC est très approprié pour la spécification temporelle et séquentielle de scénarios arbitraires dans la spécification d'un protocole. MSC peut être traduit sans trop de problème en un langage de programmation orienté objet gérant les priorités. Il faut savoir aussi que MSC ne possède pas son propre langage de spécification comme le fait SDL. On peut prendre cela comme un avantage ou un inconvénient mais nous pouvons affirmer que c'est un langage assez neutre pour l'implémentation d'un protocole.

System Description Language

La structure de SDL

Le SDL est un langage de description graphique pour la modélisation de systèmes. Il permet de spécifier et le comportement, et la structure du système.

SDL est un langage formel avec une sémantique bien définie. Donc, il est possible de transformer directement une représentation SDL en code source. SDL est donc directement comparable à UML.

SDL se base sur le principe de machines d'états qui fonctionnent de façon parallèle et auxquelles on a rajouté des timers. Les transferts asynchrones des signaux servent de mécanisme de communication entre les machines d'états qui fonctionnent sur des canaux. SDL offre aussi une communication synchrone sous la forme d'appel de procédures distant.

La structure d'une spécification SDL est réalisée de façon hiérarchique. Il y a en principe deux éléments qui sont importants :

- Blocs
- Processus

Les blocs sont hiérarchiquement situés à un niveau plus élevé des processus. Les plus hauts blocs d'une spécification se nomme un système. Chaque bloc est indépendant et est exporté parallèlement à tous les autres blocs. La synchronisation a lieu au moyen de signaux. Bien qu'il soit possible que les blocs aient leurs propres machines d'états et que les processus contiennent des blocs, il est habituel d'utiliser des blocs à la structuration hiérarchique du système, ce qui permet aussi de décrire une part du comportement du système. Les blocs et les processus sont appelés des "agents". La figure suivante illustre ces propos:

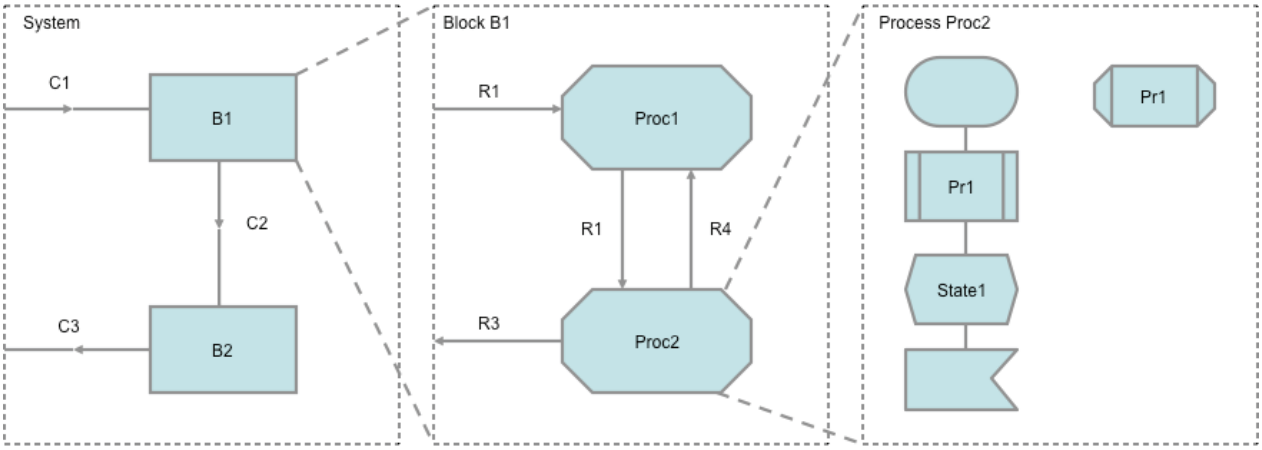


Figure 41: The structure of SDL

Agents

Comme déjà mentionné, un agent est soit un bloc, soit un processus. Les blocs sont de nature purement organisationnelle et l'élément vraiment important est donc le processus. En rapport avec les processus, les déclarations suivantes sont très importantes :

- Les processus sont exportés par une tâche qui leur est propre et possèdent leur propre zone mémoire. Cela réduit la susceptibilité d'avoir des erreurs et augmente la stabilité des systèmes qui sont spécifiés en SDL.
- Les processus communiquent mutuellement au moyen de signaux. Des signaux peuvent aussi posséder des paramètres qui transportent des informations utiles
- Les processus peuvent être organisés dans d'autres processus ainsi que dans des procédures. Généralement, il est préférable d'utiliser les procédures. Les procédures sont exportées de manière synchrone, contrairement aux processus, dans la mémoire d'un processus. Ils peuvent aussi recevoir des paramètres.
- Les processus sont responsables de décrire le comportement. Ce dernier est spécifié au moyen de machines d'états. Une machine d'états se compose d'états et de transitions. Un état attend l'arrivée d'un événement afin de passer à l'état suivant au moyen d'une transition. Pendant les transitions, des actions qui modifient le comportement du système peuvent arriver. Ces actions sont les suivantes :

  - **Tâche** : la tâche se compose d'une attribution ou d'un texte informel.
  - **Output** : une sortie produit un signal d'un certain type. Un signal est composé de paramètres qui seront envoyés à la cible.
  - **Set** : un set est exporté dans une tâche et commence un timer.
  - **Reset** : un reset est exporté dans une tâche et désactive un timer.
  - **Create Request** : des processus peuvent produire d'autres processus. Un create request peut créer une instance d'un processus et l'équiper de paramètres si nécessaire.
  - **Procedure Call** : les processus peuvent appeler d'autres processus en leur passant des paramètres, si nécessaire. Ces processus sont exportés dans la mémoire du processus appelant.
  - **RPC** : les processus peuvent aussi appeler d'autres processus qui tournent dans la mémoire d'un autre processus. On nomme ceci un remote procedure call.

Ce qui est expliqué ci-dessus est symbolisé à la figure suivante :

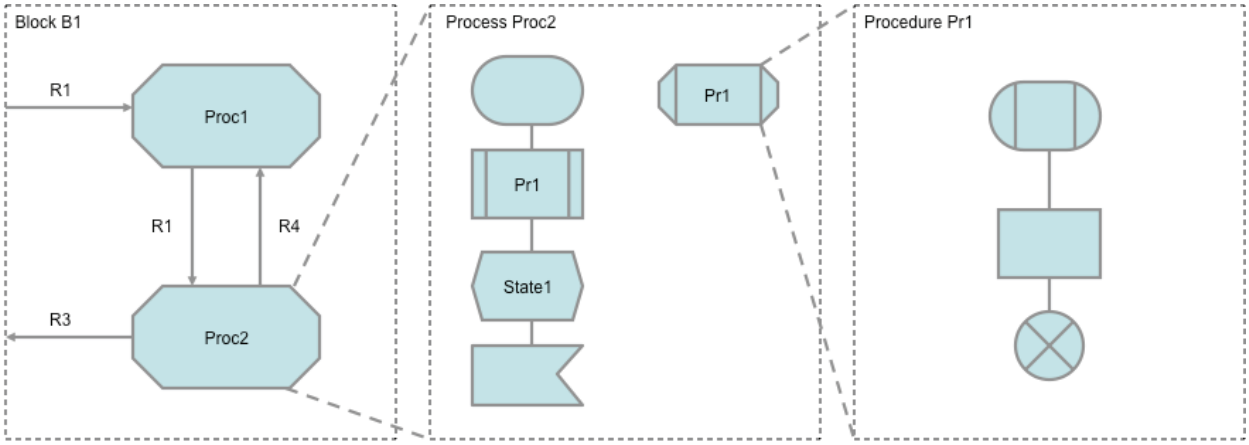


Figure 42: Processes and procedures

Communication

La communication entre agents a lieu, dans SDL, au moyen de signaux. Pour ce faire, l'agent qui envoie et celui qui reçoit doivent parler sur le même canal. Par canal, on sous-entend le medium qui permet de transmettre le signal sans retard. Chaque canal est spécifié par la liste des signaux qui peuvent être transférés dans leurs descriptions respectives. Les agents sont liés sur ce que l'on nomme des barrières sur le monde extérieur. Les barrières sont nécessaires pour permettre la liaison entre les différents diagrammes d'une spécification. Sur les barrières, il est possible de visualiser des machines d'états qui montrent quels canaux peuvent communiquer ensemble. Il en va de même si l'on définit les voies de communication indirectement de façon hiérarchique, au moyen de blocs. Les diagrammes suivants montrent de ces canaux et de ces barrières :

La figure suivante montre la communication entre les deux différents processus :

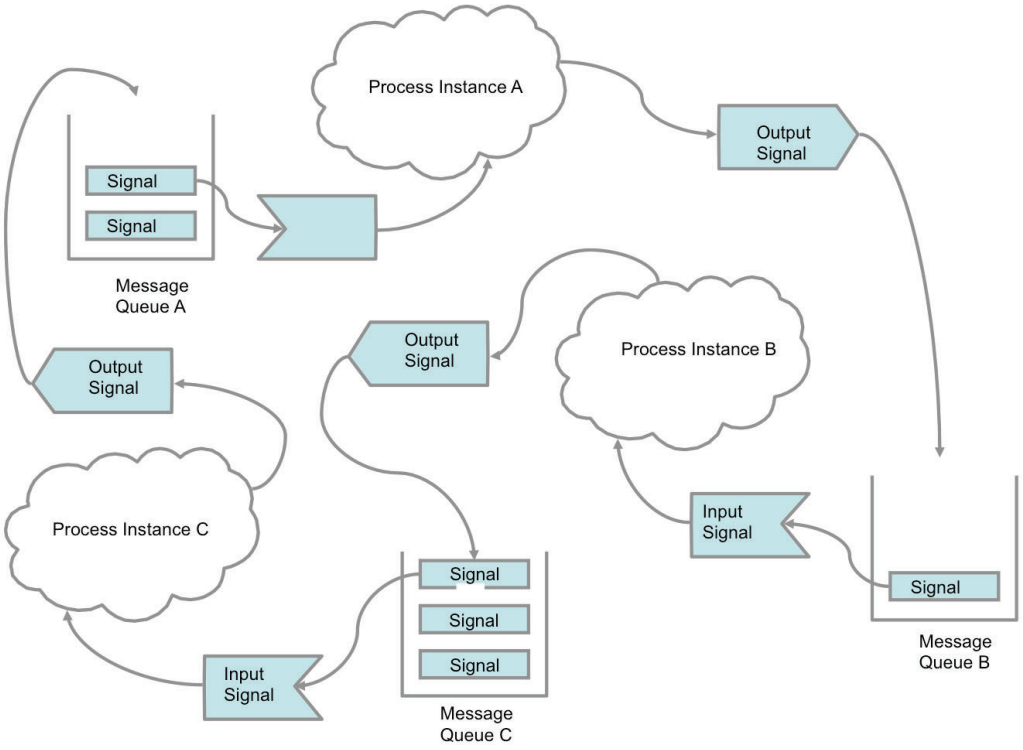
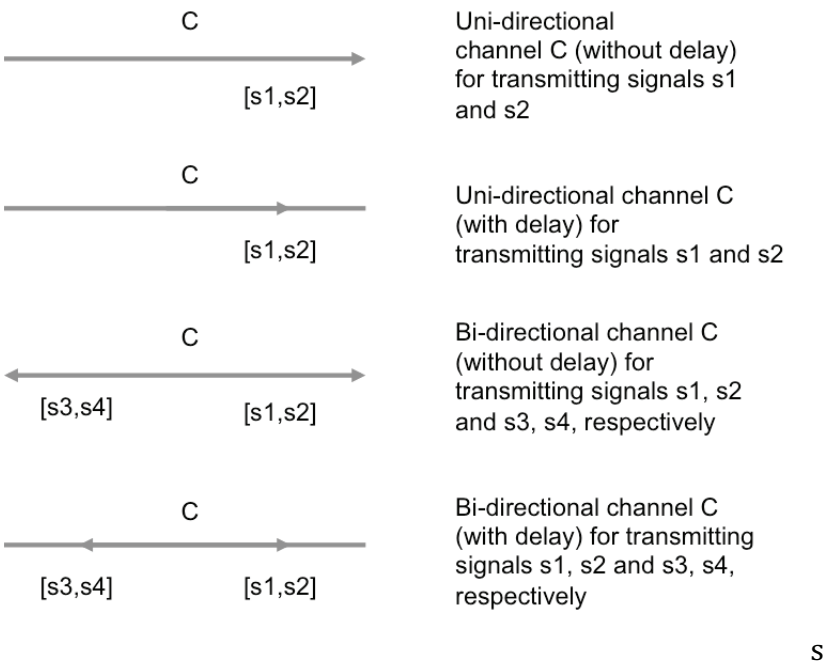


Figure 43: Communication between processes

La figure suivante montre la représentation canaux et signaux :



s

Figure 44: Channels and signals

Les éléments suivants sont à noter :

- Les voies de communication entre les blocs sont toujours appelées "canal".
- Les voies de communication entre les processus sont toujours appelées "routes de signaux".
- Les routes de signaux ne sont jamais retardées.
- Les canaux peuvent être ou ne pas être retardés.
- Une voie de communication bidirectionnelle est en fait composée de deux voies de communication unidirectionnelles.

La figure suivante montre un exemple d'un diagramme d'interaction. Ce genre de diagramme précise de quelle façon les blocs et les processus communiquent entre eux. Les interfaces de communication externes entre bloc et processus sont aussi nommées "gates".

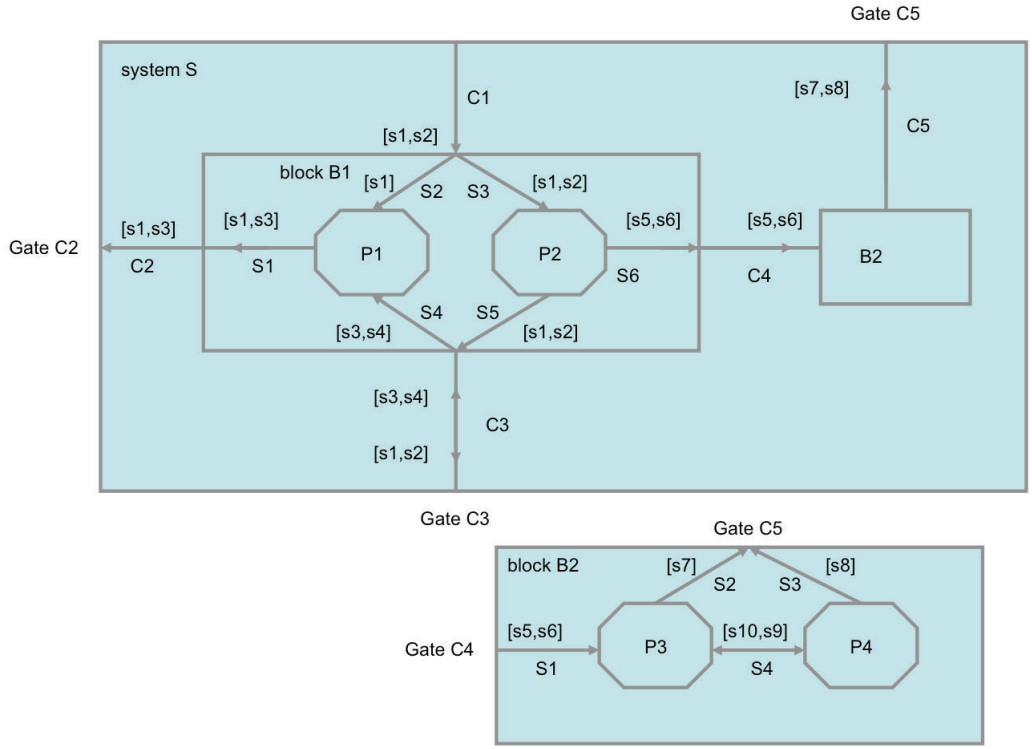


Figure 45: Interaction diagram

Représentation graphique de SDL

Dans cette section, les différents symboles graphiques de SDL ainsi que leur importance seront présentés. Le diagramme suivant montre les symboles qui sont utilisés pour la représentation des blocs et des processus :

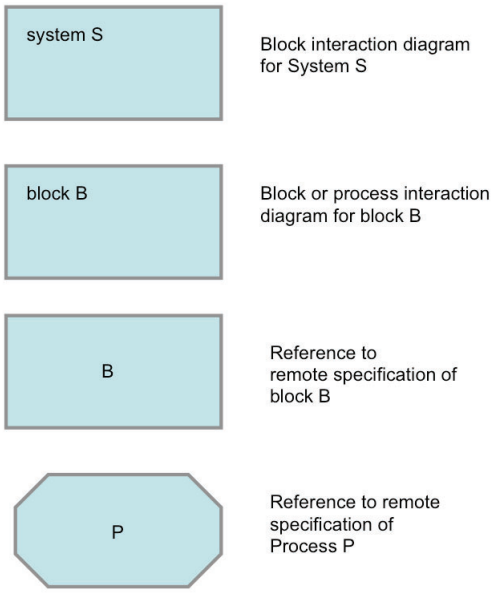


Figure 46: Symbols for blocs and processes

Dans SDL, il est aussi possible de définir des blocs ou des processus comme étant de nouveaux types. Ceux-ci pourront alors être référencés (importés) et instanciés. Le diagramme suivant montre les symboles qui deviennent utilisables :

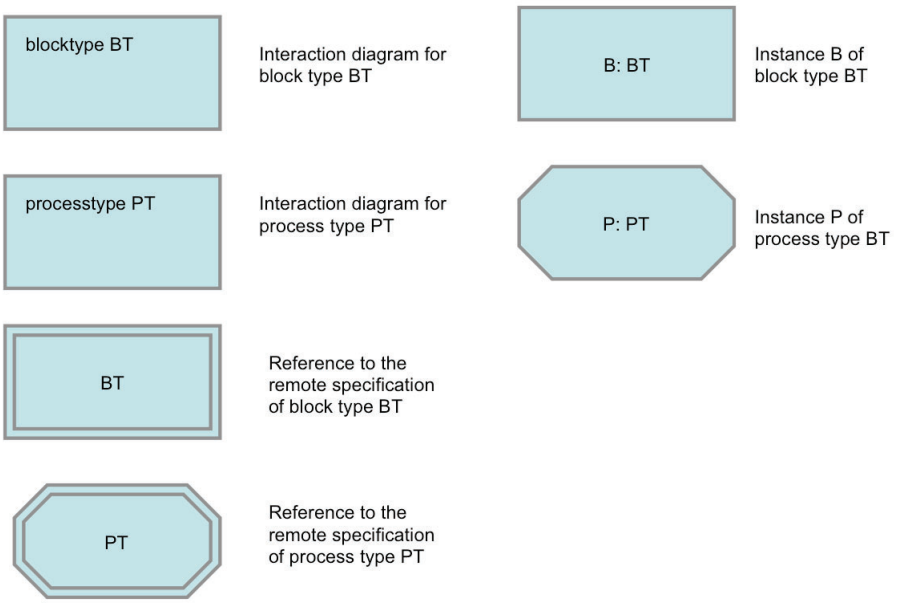


Figure 47: Symbols for bloc-types and process-types

Les symboles que nous avons traités précédemment servent à la hiérarchisation de la structure. Ceci permet de la rendre organisationnelle. Les symboles qui servent à la description des machines d'états vont donc être présentés. Remarquons une fois de plus qu'une machine d'états se compose d'un certain nombre d'éléments qui sont toujours immédiatement développés : condition initiale, transition (qui peuvent être exprimée comme étant différentes actions) et la conclusion du prochain état (ce qui devient l'état initial de l'état suivant). Le diagramme suivant clarifie cette technique :

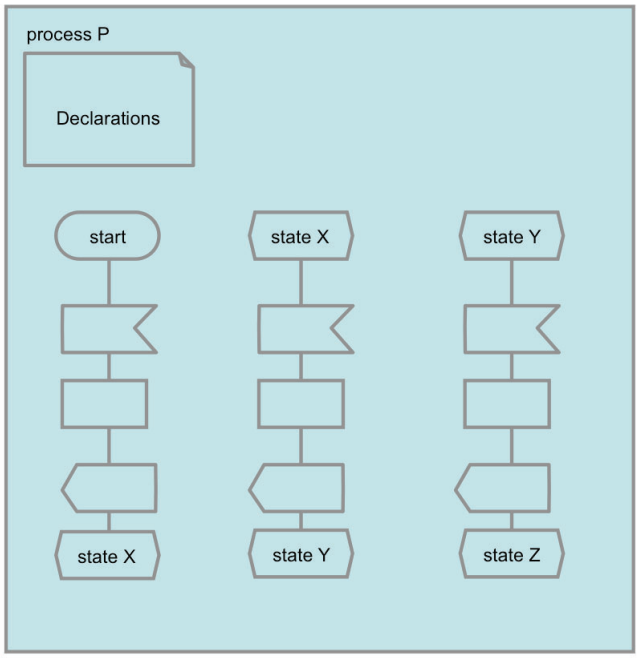


Figure 48: Basic state machine structure

Le diagramme suivant montre les éléments qui sont utilisés pour la représentation des machines d'états.

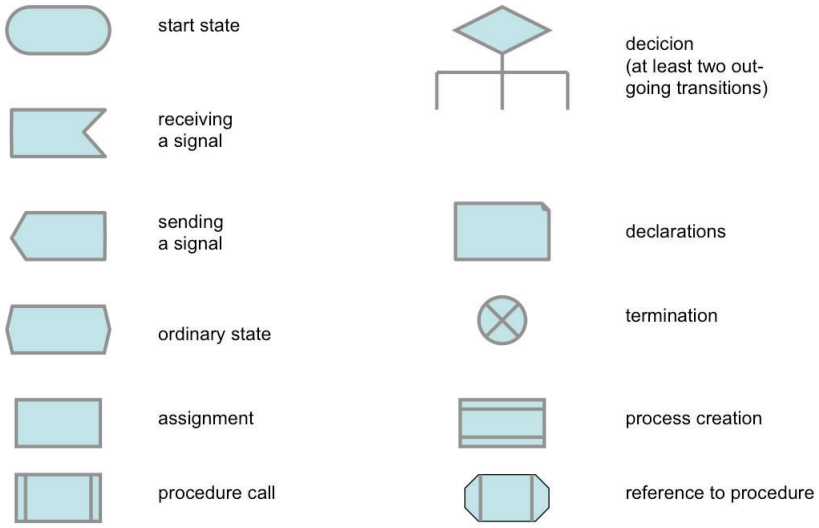


Figure 49: Symbols for the representation of state machines

En conclusion, la figure suivante montre un exemple d'une machine d'états transitions d'un processus.

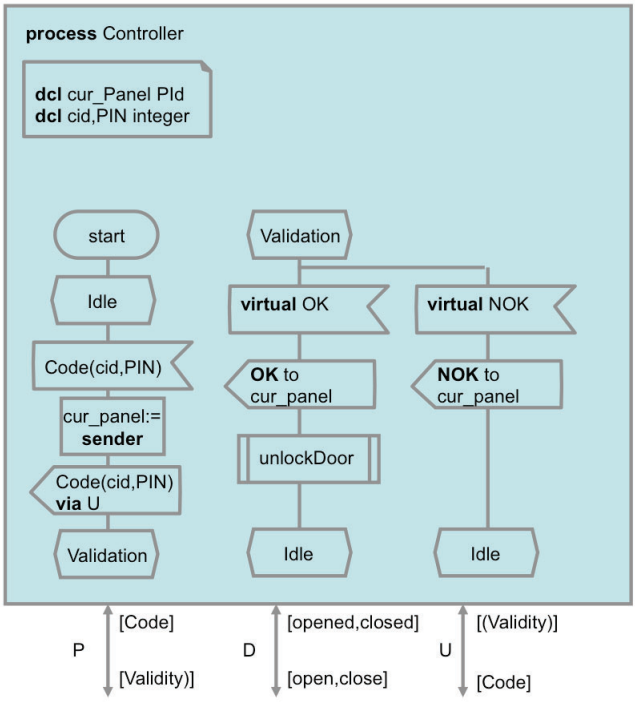


Figure 50: A simple state machine

Des processus peuvent aussi appeler des procédures. Les procédures sont définies pour transporter en lieu sûr et structurer des actions périodiques. Les procédures ressemblent énormément aux flowcharts, qui sont très semblables aux machines d'états des processus. Le diagramme suivant montre les symboles qui sont utilisés dans de tels flowcharts.

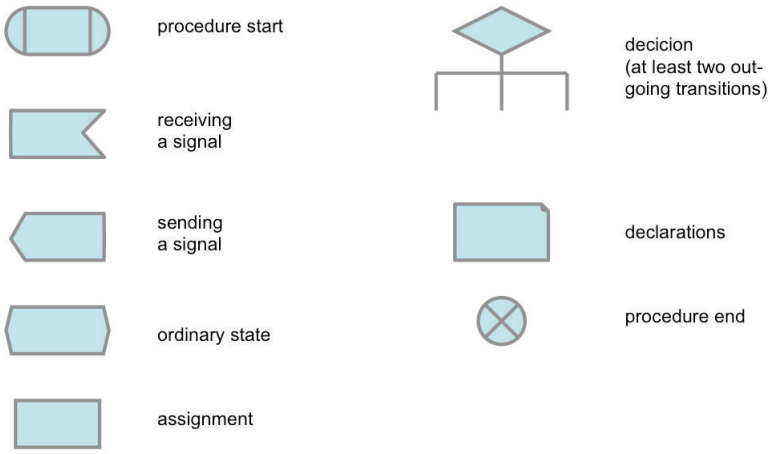


Figure 51: Symbols for the representation of procedures

Pour finir, dans le prochain diagramme, nous allons encore préciser un autre exemple de procédure :

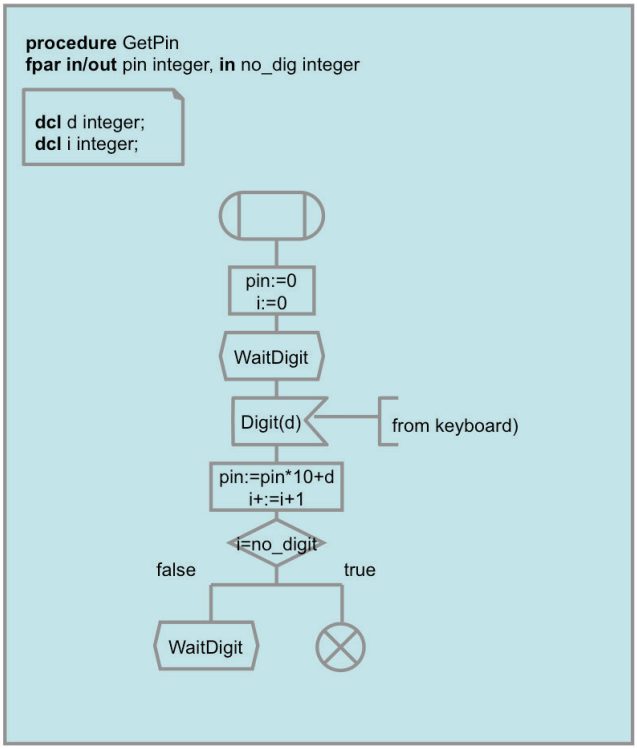


Figure 52: A simple procedure

Données

SDL a aussi la possibilité de décrire ses propres données. Ces dernières comprennent les types de base ainsi que la possibilité de fournir des types complexes. La liste suivante énumère les types de base qui sont inclus :

- integer
- real
- natural
- boolean

- character
- duration
- time
- charstring
- PId

Ci-après sont énumérés les types complexes :

- array
- struct

L'image suivant montre un exemple de définition et d'application d'un type de données définies par l'utilisateur :

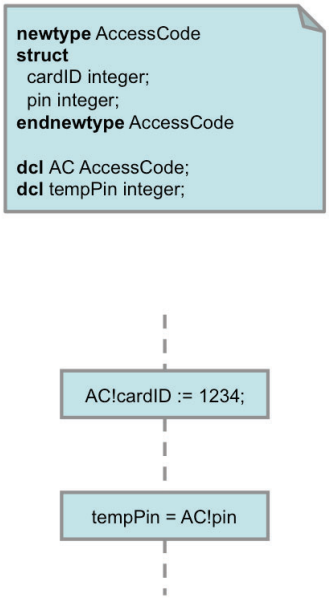


Figure 53: Data types

Les types de données ne seront pas abordés de façon détaillée dans ce cours.

Pensées finales

SDL est un langage de spécification très vaste et qui est approprié pour spécifier quelque système que ce soit, y compris les systèmes temps réels. SDL est nettement optimisé dans sa nouvelle version et nous remarquons que la différence avec UML devient toujours plus petite. Dans le monde de la spécification de protocole, SDL est toutefois comme auparavant, l'instrument de la spécification. Mais il est aussi clair que dans quelques années UML sera le remplaçant du SDL.