CONSERVATOIRE NATIONAL DES ARTS ET METIERS
PARIS

MEMOIRE

POUR L'EXAMEN PROBATOIRE

en

Ingénierie des Systèmes d'Information

par

Pierre EVEN

« Le langage SDL et son domaine d'emploi »

Soutenu le

12/01/2006

JURY

Président : Mr Dewez

Membres :

Remerciements

Je tiens à remercier Mme Graf et Mme Ober pour leur disponibilité et leurs avis éclairés. Leurs indications m'ont permis d'orienter mes recherches et de gagner des heures précieuses.

Je remercie également Mr Dewez et les membres du jury pour leur temps.

Et enfin, un grand merci à Virginie pour m'avoir soutenu et encouragé tout au long des deux derniers mois.

Table des matières

Introduction

Le « Langage de Description et de Spécification » (LDS/SDL, Specification and Description Language) est un langage formel, orienté-objet défini dans la recommandation Z-100 de l'Union Internationale des Télécommunications (UIT/ITU). Ce langage est destiné à la modélisation de systèmes complexes, événementiels, interactifs et temps réel pouvant comprendre plusieurs activités simultanées et communiquant par le biais de signaux discrets.

L'UIT (ex-Comité Consultatif International Télégraphique et Téléphonique, CCITT) est un organisme sous l'égide des Nations Unies qui propose des solutions pour la conception et la mise en œuvre de réseaux et services de télécommunications à travers le monde. Sa branche UIT-T est responsable de la rédaction de normes standardisées pour harmoniser les ressources et les pratiques liées à la conception des réseaux et services.

Le langage SDL est né du besoin de spécifier et décrire clairement les systèmes de télécommunication de tout type. Il affirme la volonté d'harmonisation de l'UIT-T en permettant une représentation graphique ou textuelle basée sur des concepts uniques et formalisés. Quelque soit l'implémentation finale, il est possible de modéliser un système au travers de concepts indépendants du système. Ainsi, le langage SDL propose d'assister la formulation des spécifications, de faciliter la conception, d'offrir un terrain favorable à la validation et l'implémentation de projets informatiques en temps réel.

Il a été conçu pour modéliser des systèmes complexes, distants, hétérogènes, et nécessitant une architecture à base d'automates finis étendus. Les systèmes étant bien souvent une juxtaposition de nombreux éléments atomiques distribués sur plusieurs réseaux homogènes ou hétérogènes et construits avec des technologies différentes. Finalement, seul un langage abstrait pouvait admettre la souplesse nécessaire pour décrire différents systèmes de ce type dans leur entièreté.

La demande pour les programmes embarqués, temps réel et leur vérification sémantique est très forte. La généralisation des produits issus ou liés aux télécommunications, et l'assistance des tâches par ordinateur laissent imaginer qu'il y aura de plus en plus de systèmes embarqués critiques qui nécessiteront une conception éprouvée. Une erreur non détectée à la phase de conception d'un projet informatique se paye au centuple par la suite.

Les projets reposant sur des systèmes événementiels, embarqués ont un niveau de complexité élevé. Les méthodes d'évaluation des coûts (ex. COCOMO) distinguent ces projets par leur non déterminisme et leur nature critique. En effet, si une application de gestion se fige, un utilisateur appelle l'administrateur pour le relancer. Ceci est simplement impensable pour le système de freinage d'un avion.

Le langage SDL se présente comme une réponse à cette demande, et semble déjà remplir ce rôle dans le domaine des télécommunications. Mais depuis 1976, il est resté très peu médiatisé en dehors de la sphère des télécommunications. Nous sommes en droit de nous demander s'il trouvera un jour son chemin vers d'autres domaines ?

Aussi, SDL n'est pas le seul langage de modélisation graphique. Le langage « Langage de Modélisation Unifié » (LMU/UML, Unified Modeling Language) est actuellement la

référence en terme de modélisation graphique et tente d'entrer en compétition avec SDL au sein des télécommunications.

Après trente années d'existence et d'évolution constante, nous pouvons nous demander si le langage SDL est arrivé à maturité, et quelle direction il prendra dans l'avenir ? Plus encore, est-ce qu'il saura répondre un jour aux problématiques hors du domaine des télécommunications ? Et finalement, pourra-t-il résister à l'avènement en force du langage UML ?

Nous allons passer en revue les concepts du langage SDL, son historique, et évaluer les forces et faiblesses respectives de SDL et UML afin de projeter ce qui, à notre avis, serait une solution optimale pour l'utilisation de l'un ou l'autre de ces langages.

Chapitre 1 - La norme SDL

Le langage SDL est avant tout une norme établie par l'UIT-T. En tant que tel, il faut le considérer comme une recommandation de bonne pratique pour décrire et spécifier sans ambiguïté des systèmes. Pour remplir cette fonction, la recommandation concernant SDL définit un certain nombre d'éléments et leurs comportements. Ces choix reposent sur une vision singulière de la façon d'élaborer des systèmes.

Le SDL fournit des concepts pour la description non seulement de la structure, mais aussi du comportement et ainsi que des données d'un système. La structure est fondée sur la décomposition hiérarchique et les types de hiérarchies. La description du comportement est fondée sur les machines à états finis étendus communiquant par message. Quant à la description des données, celle-ci est fondée sur les types de données, d'objets et de valeurs.

Afin de mieux cerner l'enjeu de la modélisation avec SDL, nous aborderons les principes fondamentaux de la recommandation, puis nous étudierons les concepts élémentaires du langage. Une modélisation SDL d'une calculatrice simplifiée est disponible dans l'Annexe 3 afin d'illustrer le propos.

Nous verrons ici la dernière version de SDL décrite dans la recommandation Z-100, à savoir SDL-2000.

1.1 Principes fondamentaux

SDL est un langage de description et de spécification dédié aux systèmes technologiquement évolués : systèmes temps réel, distribués, orientés événements. Habituellement, ce langage sert dans les systèmes de télécommunication, mais également dans l'aérospatial et dans les systèmes distribués, critiques. Pour répondre à des besoins pointus le langage SDL se fonde sur des notions clés qui lui accordent une souplesse dans la description tout en préservant la robustesse du système décrit.

1.1.1 Automates finis

Un automate fini, aussi appelé machine à états finis, est une machine abstraite utilisée en théorie de la calculabilité et dans l'étude des langages formels. Un automate est constitué d'*états* et de *transitions*. Son comportement est dirigé par un signal fourni en entrée : l'automate passe d'état en état, suivant les transitions, à la lecture de chaque signal reçu. [WIKI 2005]

En informatique, ce type de machine sert pour représenter des systèmes avec un nombre d'états distincts finis. Des automates aussi courants que la machine à café ou le guichet de banque illustrent parfaitement ce mode de fonctionnement. En effet, à la demande d'une personne (signal fourni en entrée), l'automate passe à un nouvel état pour traiter la demande. Les transitions possibles sont limitées et fonction de l'état dans lequel il se trouve.

Dans le cas du guichet bancaire, ce dernier attend patiemment un client, jusqu'à l'insertion d'une carte. À ce moment, il passe à l'état de vérification de la carte avant de, soit la rejeter, en cas d'invalidité, soit de demander le code PIN de l'utilisateur. Personne ne pourra insérer une autre carte tant que la première sera encore dans la machine.

La logique des automates finis est très utilisée en informatique pour représenter le comportement des systèmes temps réels événementiels. Elle garantie la solidité du système par des règles strictes et des opérations exhaustives, permettant notamment une validation de la couverture du graphe.

1.1.2 Langage formel

Nous désignons par langage formel un mode d'expression plus formalisé et plus précis que le langage naturel. Un langage formel se compose d'une syntaxe stricte et éventuellement d'une sémantique sous-jacente. [WIKI 2005]

Le langage SDL est un langage formel car il possède une syntaxe stricte, ainsi qu'une sémantique bien définie. Ces concepts sont définies en profondeur dans plusieurs recommandations de l'IUT-T.

Cet aspect du langage SDL permet d'automatiser la validation du code, en garantissant l'exactitude par rapport à la recommandation. De surcroît, ceci rend possible l'export automatique du code SDL vers un ou plusieurs langages exécutables.

1.1.3 Représentation graphique et textuelle

Le langage SDL possède une représentation graphique ainsi qu'une représentation textuelle. Il existe une équivalence entre la modélisation graphique et la transcription textuelle. De fait, chaque élément syntaxique possède une représentation symbolique et un équivalent graphique.

Le mode graphique de SDL reprend la modélisation état/transition en incorporant les éléments syntaxiques et sémantiques propres à SDL. La lisibilité du code en devient plus aisée et permet de conceptualiser simplement des systèmes complexes.

Le mode textuelle est certes moins lisible mais il a la propriété d'être unique et séquentiel, et donc facilement identifiable par un robot dans le cadre d'une validation.

1.1.4 Procédures et variables distantes

Pour répondre à une problématique de contexte distribué et multi processus, le langage SDL permet l'accès à des procédures et variables distantes. Par distante, il faut comprendre hors du contexte d'exécution courant.

Les systèmes complexes décrits par SDL réagissent à des signaux venus soit d'un processus interne, soit de l'environnement du système. Les procédures et variables distantes sont exportés afin d'être accéder en dehors de leur domaine de définition.

Ainsi, il est possible de permettre à un processus l'accès à une ressource ne lui appartenant pas via une procédure ou variable mise à disposition par un autre élément de son système ou d'un autre système SDL.

1.1.5 Processus concurrents

Dans un système où différents processus s'exécutent en même temps, des conflits inter-processus peuvent éclore lors de l'accès à une ressource partagée. La prévention et la gestion de ses conflits sont de toute importance pour fiabiliser et optimiser ces systèmes.

Dans le langage SDL, cette problématique est en partie résolue par les machines à états finis. Chaque machine à état simple (non composite) ne peut avoir qu'un état à la fois et ne peut entreprendre qu'une transition à la fois.

Par ailleurs, le concepteur du système peut choisir le mode d'exécution des différentes machines à états en associant ces automates soit à un *processus* SDL dont les éléments constitutifs s'exécutent en alternance, soit à un *bloc* dont les éléments évoluent en parallèle.

1.1.6 Langage Objet

Dans la programmation objet, la structure des entités du domaine passe par des classes. Un modèle de classes permet de rendre compte de la structure statique d'un système en caractérisant ses objets, leurs relations, leurs attributs et les opérations de chacun. Par extension, le paradigme objet introduit les notions de généralisation et de polymorphisme.

La généralisation est une relation hiérarchique entre une classe, la super-classe, et une ou plusieurs variantes de cette classe, les sous-classes. On dit que chaque sous-classe hérite des propriétés de sa super-classe. On dira également qu'une sous-classe est une spécialisation de sa super-classe, dans le sens qu'elle étend la définition de celle-ci. Ceci permet de structurer la description des objets et réduire le nombre de lignes de code en facilitant la réutilisation de concepts.

Le polymorphisme permet de redéfinir une opération dans une sous-classe alors que celle-ci est déjà définie dans la super-classe. Ceci apporte une plus grande flexibilité dans les opérations tout en garantissant la cohérence des structures de classe via l'héritage.

Dans le langage SDL, l'instance (d'un type) est un objet, et le type correspond à la notion de classe.

Un système SDL consiste en des instances. La classification des entités en classes et sous-classes est représentée par des types et sous-types d'instances. Par ailleurs, le langage SDL permet la redéfinition d'opérations si celles-ci sont définies comme virtuelles. Ainsi, la généralisation et le polymorphisme sont tous deux pris en compte par SDL.

En tant que langage objet, SDL gagne en clarté, en réutilisation du code mais également en portabilité vers les langages objets exécutables.

1.1.7 Evolutions du langage SDL

Le développement du langage SDL a débuté en 1972 au sein du CCITT. Un groupe de recherche constitué d'une quinzaine de personnes représentant différents pays et compagnies des télécoms (Bellcore, Ericsson, Motorola) entamaient des recherches pour mettre au point un langage de spécification standard pour l'industrie des télécoms. Sa première version fût achevée en 1976, suivie par des mises à jours tous les 4 ans jusqu'en 2000, année d'édition de la version actuellement reconnue de la recommandation Z-100 : SDL-2000.

Les changements apportés dans chaque nouvelle version se voulaient une réponse à l'évolution du monde informatique, mais également pour prendre en compte le retour d'expérience des utilisateurs et ouvrir le langage à d'autres cas d'utilisation. Une tendance constante a été de favoriser l'aspect objet du langage.

Il est intéressant de noter que la version 2004 n'est jamais parue. Une réponse possible à cet état de fait serait que le rythme et la nature des changements apportés par chaque nouvelle version étaient trop importants pour les concepteurs de logiciels. En effet, il faut savoir qu'aucune recommandation n'a été portée dans son intégralité dans un logiciel et qu'à l'heure actuelle la version la plus courante de SDL reste la version de 1996.

Une autre éventualité serait l'intérêt croissant et généralisé pour le langage de modélisation UML, y compris dans les télécommunications, résultant dans l'abandon du standard SDL.

1.2 Concepts du langage

Un système SDL se compose essentiellement d'une collection de machines à états finis s'exécutant en parallèle. Ces machines sont indépendantes mais reliées par leur appartenance à une même structure : le système.

La structure du système se divise hiérarchiquement pour être plus compréhensible et permettre de distinguer les activités. Chaque élément structurant dispose de moyens de communications pour interagir ou transmettre des signaux d'un élément vers un autre. Ces signaux sont issus ou à destination des machines à états qui reflètent le comportement de la structure englobante.

Le langage SDL permet de décrire aussi bien les types que les instances des éléments qu'il modélise. Les définitions de types indiquent les propriétés qui seront partagées par toutes les instances de ce type.

Les éléments du langage SDL peuvent être catégorisés en quatre groupes : structure, communication, comportement et données. (cf. Figure 1).

Groupe	Instance (type correspondant)	Type
Structure	Système (system), bloc (block), processus (process), état composite (bloc ou process), procédure (procedure)	System, block, process, procedure
Communication	Canal (gate), connexion (gate), signal (gate), procédure distante (remote procedure)	Gate, remote procedure, remote variable (pas d'instance)
Comportement	Machine à états finis	-
Données	-	Types de données (prédéfinis ou définis par l'utilisateur)

Figure 1. Concepts du langage SDL-2000 par instance et par type.

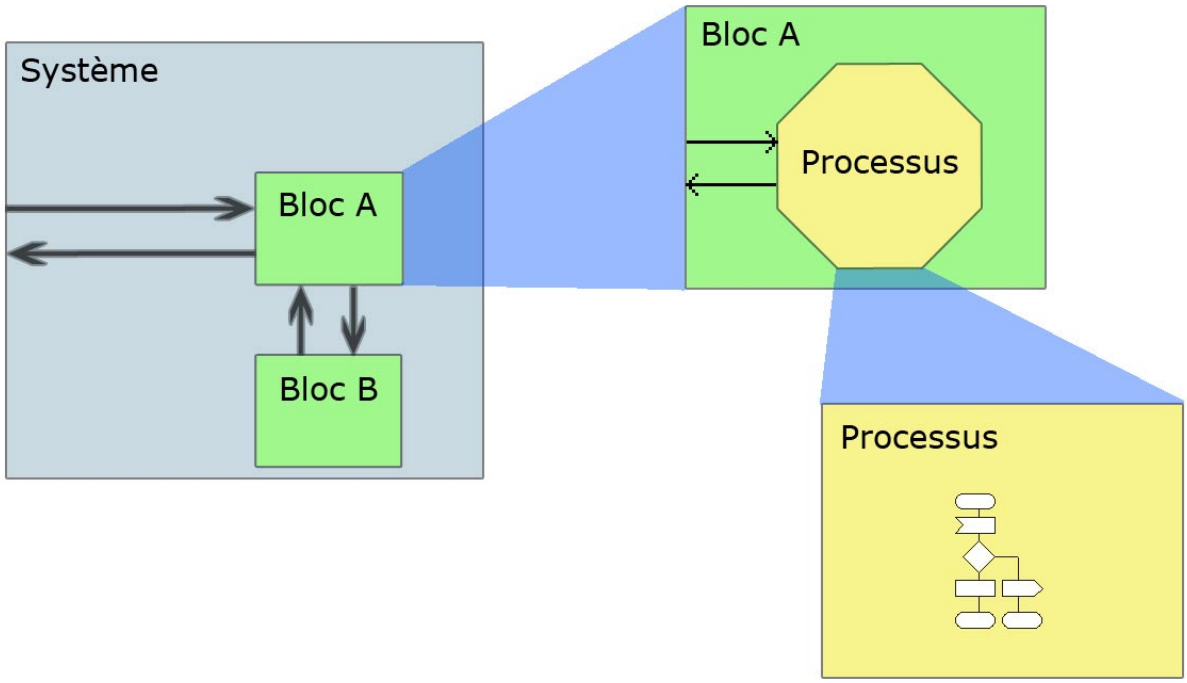


Figure 2. La hiérarchie imbriquée de SDL

1.2.1 Structure

La structure comprend les éléments : *système*, *bloc*, *processus*. Ces entités sont toutes dérivées du concept global *d'agent* apparu dans SDL 2000 pour harmoniser les définitions des éléments de structure.

« Un *agent* se caractérise par des variables, des procédures, un automate à états et des ensembles d'agents contenus. » [SDL 2000]

Nous verrons ci-après l'ensemble des *agents* du langage SDL et les *packages*, regroupement de définitions en librairies.

1.2.1.1 Le système et son environnement

Le *système* est la première entité structurante de la modélisation SDL. Il marque la frontière entre le système décrit et son environnement. A fortiori, il permet de distinguer tous les éléments du système et définit les canaux de communication pour les atteindre.

L'*environnement* du *système* n'est pas neutre. Il peut communiquer avec les éléments contenus dans un *système* par le biais de signaux empruntant les canaux pré définis.

Un *type de système* peut être créé de la même manière qu'un *système*. Celui-ci correspond alors à une *description du système* pouvant être instancier.

Un *système* peut contenir des *blocs*, des *processus* et des définitions de types. Un *système* est le *bloc* le plus à l'extérieur, contenant l'ensemble de l'univers représenté.

1.2.1.2 Bloc

« Le *bloc* est un *agent* contenant un automate à états et éventuellement, un ou plusieurs *processus* ou *blocs*. L'automate à états d'un *bloc* est interprété en parallèle avec ses *agents* contenus. » [SDL 2000]

Un *bloc* peut également contenir des définitions de types, mais il ne possède pas de variables.

1.2.1.3 Processus

Contrairement au *bloc*, un *processus* peut définir des variables locales ou distantes. Par ailleurs, l'automate à états d'un *processus* est interprété en alternance avec ses agents contenus.

Les *processus* sont les éléments principaux du langage SDL. Un *processus* dispose d'un espace mémoire dédié, s'exécute de manière concurrente avec d'autres *processus* et communique par signaux ou procédures distantes.

Un *processus* est un élément indépendant réagissant à des stimuli. La réaction d'un *processus* à un signal lui est dictée par un automate à états.

« Seuls les *processus* introduisent des données partagées dans une spécification, permettant ainsi l'utilisation des variables du *processus* englobant. Toutes les instances dans un *processus* peuvent accéder aux variables locales du *processus*. » [SDL 2000]

1.2.1.4 Etat composite

« Un *état composite* est un état qui peut être constitué soit de sous-états interprétés séquentiellement (avec des transitions associées) soit d'agrégat de sous-états interprétés dans un mode d'entrelacement. » [SDL 2000]

Un *état composite* s'exécute soit de manière séquentielle s'il appartient à un *processus*, soit de manière entrelacée s'il est compris dans un *bloc*. Il s'identifie toujours par l'agrégation de plusieurs sous-états.

Un sous-état est un état, il peut par conséquent être à son tour un *état composite*.

1.2.1.5 Package

Un *package* est une collection de types et de définitions. Une telle collection permet de regrouper différents types et définitions pour répondre à une problématique applicative ou bien selon une vision commune. Le *package* correspond à une librairie de types et permet donc de rassembler un grand nombre de types dans une seule référence afin de simplifier leur utilisation.

1.2.2 Comportement

La majorité des langages objets considèrent que le comportement est uniquement associé avec les méthodes des objets et que celles-ci peuvent être exécutées à la demande. Or ce n'est pas toujours si évident et certains langages ont choisi d'implémenter un comportement propre à chaque objet.

SDL a choisi cette deuxième vision et décrit le comportement des objets au moyen de diagrammes d'état/transition. Les états importants sont identifiés et les événements, causes de transitions entre états, ainsi que les opérations correspondantes sont décrits.

Le comportement est décrit dans les *processus* en utilisant les machines à états. Les entités *système* et *bloc* ne forment qu'une structure statique de la spécification. Chaque instance de *processus* a un identifiant de *processus* unique. Ceci permet d'envoyer des signaux à une instance spécifique d'un *processus*.

1.2.2.1 Machines à états finis

Dans SDL, le comportement est défini par le biais de machines à états finis étendues. Le comportement est implémenté au niveau *processus* et se traduit par un graphe avec des états et des transitions.

Dans un état donné, le *processus* peut recevoir un certain nombre de signaux, et la consommation du signal entraîne l'exécution de la transition suivante et l'entrée dans un nouvel état.

Une transition trouve son origine dans un premier état, fini dans un second (possiblement le même qu'au départ), et se déclenche par des signaux entrants reçus par la machine à états. Les messages entrants se placent dans la pile d'entrée du processus.

Tous les signaux entrants reçus par un *processus* sont placés dans sa pile d'entrée en attendant d'être traités par l'automate à états finis. Ce dernier va soit les traiter, soit les sauvegarder pour un traitement ultérieur, ou simplement les rejeter. SDL implémente une gestion FIFO (First In First Out) des messages.

Lors d'une transition, la machine à états peut réaliser plusieurs actions : affectation d'une variable, création d'un processus, appel d'une procédure (locale ou distante), émission de signaux, opérations de temporisation, etc.

1.2.2.2 Procédure

Une procédure permet de décomposer la spécification du comportement en séquences d'actions liées logiquement.

Une procédure se définit localement par rapport à un processus ou globalement par rapport à un bloc, un système ou un package. Afin qu'elle puisse être exécutée par d'autres processus, celle-ci devra être exportée par le processus sous forme de procédure distante.

Les procédures utilisent les machines à états de manière similaire aux processus. L'exécution de la machine à états d'une procédure influe sur le comportement du processus contenant la procédure. Par exemple, un processus peut arrêter son exécution temporairement parce qu'une procédure attend dans un certain état.

Une procédure s'achève lorsque son automate atteint un nœud de sortie. A ce moment, les paramètres en sortie de la procédure sont retournés à l'instance appelante.

1.2.3 Communication

Le comportement précédemment décrit est tributaire des signaux et plus généralement de la communication entre éléments à l'intérieur ou avec des éléments à l'extérieur du système.

La communication passe par le biais de signaux empruntant des canaux, accès et connexions prédéfinis par le système.

1.2.3.1 Signal

Dans SDL, la communication inter processus est basée sur l'échange de messages discrets. Chaque *processus* dispose d'une pile destinée à recevoir des signaux. La réception d'un signal permet de déclencher une transition d'un automate, d'un état vers un autre.

Un signal possède un nom et éventuellement des valeurs, appelés paramètres. Il se déplace de manière unidirectionnelle ou bidirectionnelle sur des canaux passant par des connexions entre deux processus d'un système unique ou plusieurs systèmes distincts.

En plus des signaux, les *processus* peuvent communiquer par procédures ou variables distantes. Du côté serveur, une procédure ou l'accès à une variable distante sont traités comme un signal. Alors que du côté client, l'exécution est bloquée en attendant un retour de la procédure ou la valeur de la variable.

1.2.3.2 Canal

Un canal représente une route pour les signaux entre deux points terminaux (accès, connexion, *agent* ou automate à états). Un canal peut être unidirectionnel ou bidirectionnel et il a la possibilité de définir les types de signaux qu'il admet.

Un canal peut ainsi relier des agents internes du système et leurs permettre de communiquer. Il peut également relier l'automate à états (*état composite*) d'un *agent* avec l'environnement et avec des *agents* contenus.

A l'extrémité de destination d'un canal, les signaux se présentent dans le même ordre que celui des signaux à son origine. Si deux ou plusieurs signaux arrivent simultanément sur le canal, ils sont ordonnés arbitrairement.

1.2.3.3 Accès & Connexion

Un accès est un point de terminaison potentiel pour un canal. Il fait parti d'un agent, une déclaration de type d'*agent* ou bien d'un automate à états. A l'image du canal qui peut connecter à l'accès, ce dernier est unidirectionnel ou bidirectionnel et peut restreindre la liste des signaux qu'il admet.

Une connexion représente un accès implicite vers une machine à états permettant de joindre plusieurs canaux ou de séparer un canal en plusieurs. Ainsi, plusieurs messages d'origines différentes peuvent être adressés au même état, et inversement, un message peut être adressé à plusieurs états.

1.2.4 Données

Le langage SDL accepte deux types de descriptions de données, ADT (Abstract Data Type) et ASN.1 (Abstract Syntax Number One). L'intégration de ASN.1 permet le partage de données entre langages, aussi bien que la réutilisation de structures de données déjà existantes. Ci-après nous nous attacherons au type de base ADT et les éléments prédéfinis à l'usage des concepteurs.

1.2.4.1 Abstract Data Type

Les données dans le langage SDL sont principalement abordées sous l'aspect des types de données. Un type de données définit un ensemble d'éléments désignés par le terme *sorte*, et un ensemble d'opérations qui peuvent être appliquées à ces éléments de données. Les sortes et les opérations définissent les propriétés du type de données. Ces propriétés sont définies dans des définitions de type de données.

« La notation ADT est un type de données sans structure de donnée spécifiée. En revanche, il spécifie une série de valeurs, d'opérations autorisées, et d'équations à base des opérations définies précédemment. Cette approche facilite le mappage d'un type de donnée SDL avec un type de donnée utiliser dans d'autres langages de haut niveau. » [SDL 2000]

« Chaque élément de données appartient à une seule et unique sorte. C'est-à-dire que les sortes n'ont jamais d'éléments de données en commun. » [SDL 2000]

1.2.4.2 Le package *Predefined*

Le package « predefined » (prédéfini) contient une collection de définitions des types les plus courants. Voici la liste de ces types :

Types simples
• Boolean, Integer, Natural, Real,
• Character, Charstring, Bit, Bitstring,
• Octet, Octetstring,
• Time, Duration et Pid

Types évolués
• String, Powerset, Bag,
• Array, Vector

Exceptions
• OutOfRange, InvalidReference,
• NoMatchingAnswer, UndefinedVariable,
• UndefinedField, InvalidIndex,
• DivisionByZero, Empty

Chapitre 2 - SDL en action

Le langage SDL est une norme standardisée, non propriétaire, crée par l'UIT-T en partenariat avec l'industrie des télécommunications. Depuis 1972, de nombreuses entreprises du domaine des télécommunications ont pris part au développement du langage SDL afin qu'il réponde à leurs problématiques.

L'omniprésence de SDL dans ce secteur d'activité se constate encore à l'heure actuelle dans les spécifications des systèmes, et laisse penser que son développement fut fructueux. Des documentations et spécifications en SDL existent chez la quasi-totalité des acteurs du paysage des télécommunications.

Cependant, malgré des évolutions fréquentes pour améliorer, généraliser et adapter le langage SDL, ce dernier semble aujourd'hui en perte de vitesse.

Pour comprendre la position du langage SDL aujourd'hui et afin de spéculer sur son avenir, nous allons établir une cartographie de la sphère SDL. Dans cette cartographie, nous nous attacherons aux grands acteurs du langage, puis à la tentative d'ouverture du langage pour une utilisation généralisée dans les systèmes temps réels et finalement nous observerons les outils disponibles aujourd'hui pour mettre en œuvre SDL.

2.1 Les acteurs de SDL

Parmi les acteurs responsables de la création, l'évolution et la prolifération du langage SDL nous constatons quatre grandes entités, les organismes ou associations, les chercheurs en mathématiques appliqués et en informatique, les industriels en télécommunication et les concepteurs de logiciels pour la spécification de systèmes réactifs.

Cette division n'est pas exclusive dans le sens où certains chercheurs travaillent pour des industriels et participent également à la vie des organismes. Nous voulons simplement souligner les visions et intérêts qui s'affrontent et se complètent pour former le lieu de création d'un langage. Effectivement, il est intéressant de constater le nombre d'actions communes s'appuyant sur SDL, liant des chercheurs, industriels et concepteurs de logiciel.

2.1.1 Organismes

2.1.1.1 L'Union Internationale des Télécommunications (UIT)

L'UIT, anciennement CCITT, est un organisme international sous la responsabilité des Nations Unis, ayant pour mission de coordonnées les méthodes et les pratiques en matière de télécommunication.

L'UIT se divise en 3 sections :

- **UIT-R** pour les radiocommunications,
- **UIT-D** pour le développement des télécommunications,
- et **UIT-T** pour la normalisation des télécommunications.

SDL est l'œuvre du groupe de travail n°17 de l'UIT-T et correspond aux recommandations Z-100 à Z-110.

Les normes MSC et TTCN sont également définis dans la série Z des recommandations. Ces deux écritures complètent SDL en permettant de rédiger des cas d'utilisation, des séries de tests et une validation d'un système.

L'UIT-T est resté très actif dans le développement de SDL, jusqu'en 2000, en publiant tous les quatre ans une nouvelle version du langage. Ils sont garants de l'indépendance et de l'évolution du langage pour remplir son rôle de spécification formelle de systèmes complexes.

Cela dit, il n'est pas responsable de la promotion du langage, ni même de la synchronisation des utilisateurs. D'ailleurs, en terme de promotion, la recommandation en elle-même est très rébarbative et il faut s'armer de patience pour percer les mystères du document Z-100.

2.1.1.2 Le SDL-FORUM

Le SDL-Forum est une association visant à réunir et faire dialoguer toutes les personnes intéressées par SDL. Ce collectif dispose d'un site web distillant des informations sur le langage et orientant les nouveaux utilisateurs.

Par ailleurs, l'association organise annuellement un colloque réunissant des acteurs du monde SDL afin de présenter des travaux, discuter des évolutions possibles du langage, ou plus généralement partager ses connaissances.

Un résumé de chaque séminaire est édité et peut être commandé sur Internet. Généralement, ces réunions abordent des sujets très pointus du langage SDL, ou des langages formels proches et s'adressent essentiellement à un publique averti.

Par exemple, le colloque de juin 2005 s'est attaché à des sujets aussi variés que :

- « IF: A Validation Environment for Real-time UML and SDL Models »
- « Semantics of Message Sequence Charts »
- « Modeling, Verifying and Testing Mobility Protocol from SDL Language »
- « Consistency Checking of Concurrent Models for Scenario-Based Specifications »
- « SDL Code Generation for Open Systems »

Force est de constater que l'affluence du forum en ligne baisse depuis quelques années. Les débats sur le forum sont rares en comparaison des annonces de « request for papers » ou d'événements destinés aux spécialistes. Ces deux facteurs soulignent la faible médiatisation du langage et laisse entrevoir un fossé entre les utilisateurs actuels est les utilisateurs potentiels de demain.

2.1.1.3 Le Réseau National de Recherche en Télécommunications

« Ce réseau a pour objectifs de dynamiser l'innovation en favorisant la confrontation entre les avancées technologiques et les besoins du marché en facilitant le transfert technologique vers les entreprises et d'accompagner l'ouverture des marchés à la concurrence et l'évolution du rôle du CNET dans la recherche publique. Le RNRT offre ainsi à la recherche en télécommunications un espace ouvert, créé pour inciter les laboratoires publics, les grands

groupes (industriels et opérateurs) et les PME à se mobiliser et à coopérer autour de priorités clairement définies, pour conduire des projets avec le soutien des pouvoirs publics. » [RNRT]

Cet organisme gouvernemental pour le développement des télécommunications s'est récemment (entre 1999 et 2001) intéressé au langage SDL dans le cadre du projet CONVERGENCE. Ce projet avait pour but « d'accroître la compétitivité des entreprises françaises dans les secteurs industriels tels que les Télécoms ».

Persuadé du bénéfice que pouvait apporter la rigueur du langage SDL dans la construction de modèles exécutables et de l'apparition de nouvelles opportunités pour SDL-2000, le RNRT a commandité le projet Convergence avec la participation des chercheurs de l'IRISA, l'industriel Alcatel et du concepteur Telelogic. Devant l'avènement du langage UML, ce projet visait à établir un état de l'art d'UML et de faire des propositions d'évolutions pour les deux langages SDL et UML. Le bilan de cette opération fut édité en novembre 2001.

Parmi les apports de ce projet :

- « Pour l'UIT : contributions à la normalisation de SDL-2000, Recommandations de la série Z.100 approuvées en décembre 1999. » [Ober 2003]

- « Pour l'OMG : contributions principalement à la demande de propositions « Action Semantics » et à la préparation des RFI et RFP « UML 2.0 ». » [Ober 2003]

« Ces résultats contribuent directement à la convergence entre les 2 notations telle que définie dans les objectifs du projet. Il est à noter que les propositions de CONVERGENCE pour SDL-2000 ont été largement adoptées par l'UIT. En ce qui concerne la proposition faite à l'OMG pour le modèle sémantique du langage d'action, elle a été amendée et a abouti à une fusion avec des propositions formulées par d'autres répondants. Les travaux ultérieurs du projet ont été basés sur la soumission finale. » [Ober 2003]

« En complément à ces études, le projet a permis d'avancer sur les deux axes complémentaires, de l'outillage d'une part, et de la méthodologie d'autre part. » [Ober 2003]

Ce type d'initiative revêt des aspects techniques, politiques et économiques. En effet, le RNRT met l'accent sur la collaboration nationale pour la réalisation de projets techniquement pointus. Et ceci afin de montrer une compétence nationale, de créer une connaissance sur le territoire et en espérant donner une avance technologique aux acteurs économiques nationaux.

Au vu du bilan, ce projet semble avoir porter ses fruits. En effet, les points pris en compte dans SDL et UML, nous rapprochent d'une convergence, bien que cela ne soit pas encore le cas aujourd'hui. La valorisation de cette opération pour les entreprises françaises est difficilement quantifiable.

Un seul doute subsiste : Est-ce que les projets de ce type aide le langage SDL à se démocratiser ou, au contraire, le pousse vers une mort prématurée ?

2.1.2 Chercheurs

Il est impossible d'établir une liste exhaustive des chercheurs et groupes de recherche utilisant ou ayant utiliser SDL. Parmi ceci nous présenterons le VERIMAG en France, pour leur

implication dans le développement de systèmes embarqués et plus particulièrement leur participation active aux colloques SDL-Forum.

Notons cependant, une forte implication des pays scandinaves précurseurs dans le domaine des télécommunications.

Par ailleurs, même si l'Europe est grandement représentée dans les publications présentées aux SDL-Forum, nous retrouvons également des équipes venant du Canada, de Chine, des Etats-Unis ou encore d'Inde. Finalement, Il s'avère que SDL est une recommandation mondialement connue et reconnue dans le milieu de la recherche.

2.1.2.1 Le VERIMAG

Le VERIMAG a été créée en 1993 par le Centre National de la Recherche Scientifique (CNRS) et la société Verilog. Depuis 1997, VERIMAG est une unité de recherche du CNRS, de l'Institut National Polytechnique de Grenoble (INPG), et de l'Université Joseph Fourier (UJF), et fait partie de la fédération de l'Institut d'Informatique et Mathématiques Appliqués de Grenoble (IMAG).

VERIMAG est organisé en trois équipes :

- **Synchrone** pour les langages synchrones et systèmes réactifs,
- **DCS** pour les systèmes répartis et complexes,
- Et **Tempo** pour les systèmes temporisés et hybrides.

VERIMAG se considère comme l'un des leaders du domaine des systèmes embarqués. Et leurs travaux visent à produire des outils théoriques et techniques pour permettre le développement de systèmes embarqués de qualité maîtrisée, et ce à des coûts compétitifs.

Le VERIMAG a développé plusieurs projets en rapport avec SDL, dont certains ont été présentés à l'occasion d'un SDL-Forum. Il est d'ailleurs un acteur récurrent des SDL-Forum, notamment au travers de Mme Suzanne Graf, chercheur senior au VERIMAG.

Mme Graf a eu l'amabilité de me livrer sa vision actuelle du langage SDL :

Selon elle, SDL est un langage assez intéressant avec une sémantique formelle et des outils commerciaux pour la validation, la génération de test et la génération de code.

Les trois outils les plus connus sont ObjectGeode de Verilog, TAU de Telelogic et Cinderella par la une toute petite société danoise du même nom. Puis il y a également des outils comme ceux de PragmaDev, supportant un langage "SDL like".

Mais aujourd'hui, les vrais outils (de Verilog/Telelogic) ont quasi disparu, et les opérateurs des Telecom ont la tendance à abandonner SDL au profit de UML. Cette tendance n'est pas forcément justifiée, car UML pose énormément de problèmes. A part les logiciels proposés par les éditeurs il n'y a pas vraiment d'outils indépendants, sans compter que le standard parle peu de sémantique et ne se porte pas à la validation formelle par les outils.

Par ailleurs, la dernière version du langage SDL (SDL-2000), n'est implémentée par personne. Telelogic a sorti TAU-G2 qui est un outil UML avec une option SDL ; Il est possible

d'assimiler SDL à un profil particulier de UML 2.0. Certains diagrammes « à la SDL » font désormais partie intégrante du standard UML.

2.1.3 Industriels

Premiers intéressés par le développement de méthodes et d'outils pour améliorer la conception de systèmes complexes, les industriels participent activement au développement du langage SDL. En France, deux acteurs se distinguent par leur notoriété et leur ampleur : France Télécom et Alcatel.

2.1.3.1 France Télécom

France Télécom est l'opérateur historique français. Aujourd'hui, cette entreprise développe et commercialise des services pour les particuliers et les entreprises.

Les marques réunies sous la bannière de France Télécom sont : France Télécom (évidemment), Orange, Orange Entreprises, Equant et Oléane.

France Télécom est organisée en 5 divisions opérationnelles, 5 divisions fonctionnelles et les traditionnelles fonctions support (RH, finances, communication). Parmi ces divisions nous retrouvons la « Division Technologie et Innovation » qui intègre principalement « France Télécom Recherche & Développement », l'ancien « CNET ». [WIKI 2005]

La section Recherche & Développement est un acteur omniprésent du développement des télécommunications en France. A ce titre, il est tout naturel de voir France Télécom associé au langage SDL : au travers de sa présence aux SDL-Forums, en tant qu'acteur de la standardisation de ASN.1 et surtout exploitant du plus gros réseaux de télécommunication sur le territoire français. Ils sont évidemment à compter parmi les utilisateurs d'outils à base de SDL.

2.1.3.2 Alcatel

« Alcatel est un équipementier en télécommunications. Cette entreprise privée est le leader mondial dans la fourniture de commutateurs téléphoniques, des équipements de transport optique, des routeurs ATM et IP, des câbles de transmission sous-marins, de l'infrastructure mobile (GSM, GPRS, UMTS), des applications de réseaux intelligents, des applications de Centre d'Appel, des applications vidéo, ainsi que des satellites et des charges embarquées. » [WIKI 2005]

Alcatel est également prestataire de services, depuis la conception de réseaux jusqu'à l'exploitation de ceux ci en passant par le déploiement, l'intégration et l'installation.

Elle a également participé au projet CONVERGENCE et montre ainsi son engagement dans la recherche et le développement de nouvelles méthodes de spécification pour les systèmes complexes.

Ainsi, l'entreprise connaît très bien les problématiques des systèmes complexes, répartis, communiquant par signaux dans les équipements en télécommunication et s'intéresse de très près aux outils à base de SDL.

En effet, nous retrouvons actuellement chez Alcatel deux outilleurs de la sphère SDL, à savoir, Telelogic et PragmaDev. Le contrat liant Alcatel et PragmaDev a été officialisé par un communiqué de presse en mai 2005, stipulant que l'équipementier avait fait l'acquisition de l'outil RTDS G3 pour le développement de ses systèmes de commutation Alcatel 1000. Notons toute fois, que l'outil RTDS G3 n'est pas un outil SDL classique marquant une rupture avec la spécification originale. L'outil RTDS G3 sera décrit dans la section 3.2.2.3.

« Le support du SDL dans notre dernière version avec UML et SDL-RT au sein du même environnement homogène fait de RTDS G3 le solution la plus souple et la plus complète pour le développement des applications temps réel et embarquées. En sélectionnant notre solution, Alcatel a valorisé son existant SDL et s'ouvre au SDL-RT pour leurs futurs développements validant ainsi la vision de PragmaDev. » Commente Emmanuel Gaudin, le directeur de PragmaDev, dans ce communiqué de presse.

2.1.4 Concepteurs de logiciels

La présentation des concepteurs de logiciels qui suit provient en grande partie des informations distillées par ces mêmes entreprises. Il faut donc garder à l'esprit que les informations suivantes ne sont pas neutres et relèvent d'une stratégie de communication.

2.1.4.1 Telelogic

Sans doute le numéro un des solutions de développement à base de SDL grâce à SDL Tau et anciennement le très populaire ObjectGeode de Verilog.

Domicilié à Malmö, Suède, la société Telelogic possède des antennes dans une vingtaine de pays à travers le monde.

Parmi ces clients nous retrouvons : Airbus, Alcatel, BAE SYSTEMS, BMW, Boeing, DaimlerChrysler, Deutsche Bank, Ericsson, General Electric, General Motors, Lockheed Martin, Motorola, NEC, Philips, Samsung, Siemens, Sprint, Thales, et Vodafone.

En Décembre 1999, Telelogic annoncée l'acquisition de l'éditeur de logiciel français Verilog, auparavant détenue par Communication et Systèmes. Cette acquisition faisait de Telelogic la plus grande société du marché des logiciels de développement temps réel.

Le but avoué de cette fusion était d'unir le savoir faire des deux éditeurs pour créer une nouvelle génération d'outils à base de SDL.

Aujourd'hui, Telelogic commercialise une demi-douzaine de suites logiciel dont la suite Telelogic Tau pour la conception, le développement et les tests de systèmes informatiques. Le produit phare de cette gamme est Tau G2 qui repose essentiellement sur UML 2.0. Nous voyons par là un changement notable dans l'orientation de l'éditeur vers une promotion du nouveau langage UML. Toute fois, il existe également Tau SDL Suite, présenté comme l'outil de prédilection des développeurs de protocoles de télécommunication, mais certainement pas la solution pour toute conception de système temps réel.

2.1.4.2 PragmaDev

PragmaDev est une société française créée en 2001 par Emmanuel Gaudin, ancien consultant et formateur SDL chez Telelogic.

Cette entreprise propose une suite d'outils pour le développement de systèmes embarqués temps réel : Real Time Developer Studio. Elle cible les équipes de développement travaillant avec des Systèmes d'Exploitation Temps Réel (SETR), notamment en leur offrant des outils graphiques dans un marché où 90% des développements seraient réalisés dans un mode textuel.

Fort de son titre de lauréat du concours national du Ministère de la Recherche en 2001 pour la création d'entreprises de technologies innovantes, PragmaDev est aujourd'hui le partenaire d'acteurs clés des applications du temps réel. Quelques-uns de ses clients sont: Airbus, Alcatel, Thomson, L'armée française, Nortel Networks, Korean Telecom, et LG Electronics.

PragmaDev est aussi à la base du mouvement dissident SDL-RT. Ce langage a été crée à partir de SDL, pour les développeurs d'applications temps réel en permettant de conserver les types et sémantiques du langage C.

Les arguments principaux en faveur de SDL-RT sont la connaissance quasi universelle du langage C, son application extensive dans les développements de systèmes temps réel, liés à l'ergonomie et à la rigueur du langage formel SDL. Offrant ainsi un outil efficace pour un investissement faible et des résultats probants.

2.1.4.3 Cinderella

Cinderella fut fondée en 1995 à Copenhague, au Danemark. Cette société possède une expertise dans le domaine du développement logiciel et notamment dans les techniques de spécification standardisées.

Le PDG, Anders Olsen, est impliqué dans la standardisation du langage SDL depuis 1988 et un des acteurs principaux de la définition formelle du langage. Aujourd'hui encore, Cinderella est activement impliquée dans les travaux de standardisation de SDL, MSC et ASN.1.

Le premier produit de Cinderella, Cinderella SDL, est sorti en Avril 1998. Aujourd'hui, cet outil est utilisé par des entreprises et universités à travers le monde.

Leurs clients proviennent exclusivement des secteurs télécommunications et systèmes critiques mais ne sont pas cités.

La petite Cinderella ne semble pas avoir une réelle politique de croissance, mais plus de développement, en suivant les évolutions en parallèle de SDL et d'UML.

2.2 De l'emploi de SDL

Les acteurs de SDL se trouvent essentiellement dans le domaine des systèmes temps réel et bien souvent dans les télécommunications. Emmenés par les concepteurs de logiciels, ils

tentent d'ouvrir le langage à d'autres domaines d'emploi théoriquement propices aux langages formels.

Dans cette partie, nous allons commenter les tentatives d'étendre le champ d'application du langage hors du domaine des télécoms et ensuite nous présenterons les outils logiciels à base de SDL, pour concevoir des systèmes complexes, temps réel et communiquant.

Dans la première partie, nous fonderons notre analyse sur une publication d'Emmanuel Gaudin, PDG de PragmaDev, présentée lors du SDL-Forum 2003 [GAU 2003].

2.2.1 Ouverture du langage

En 1994, une étude comparée de douze langages de spécification [KARL 94] a été menée sur un cas concret de robot industriel. Cette étude s'appliquait au langage SDL, mais également à Esterel, Lustre, StateCharts/STD, Raise, Focus, KIV, Tatzelwurm, Deductive Program Synthesis, Spectrum, Troll, Eiffel & Modula-3.

L'étude a célébré la représentation graphique possible grâce au langage SDL et encore largement absente dans le domaine de la modélisation de systèmes temps réel. En revanche, l'absence d'une vérification mathématique précise a amoindri l'intérêt d'un langage formel et de SDL en particulier.

Finalement, les auteurs ont reconnu l'intérêt d'un langage graphique pour la structuration et la communication des spécifications, mais cela n'était pas suffisant pour plébisciter SDL. A leurs yeux, le langage manquait d'outils suffisamment matures et ils jugeaient ne pas pouvoir tirer profit de l'aspect formel de SDL. Cette étude fait écho à d'autres travaux similaires qui reconnaissent également le potentiel de SDL mais regrettent le manque d'outils techniques existants pour tirer profit de la recommandation.

L'expérience de certains a démontré que les principes fondamentaux de SDL pouvaient s'appliquer dans des cas de systèmes embarqués temps réels très variés. Notamment son approche fonctionnelle et sa représentation graphique de la structure et du comportement d'un système.

Cependant, l'implémentation des spécifications sur la cible s'est souvent avérée périlleuse. Les concepts abstraits de SDL ne correspondaient pas toujours à la réalité du système. De plus, certains concepts temps réels ne sont pas pris en charge par le SDL de la norme Z-100. Ainsi nous avons pu constater l'apparition d'outils propriétaires pour compléter les insuffisances de la recommandation d'origine. Ceci engendra un qui pro quo suffisant pour enrailler la propagation annoncée de SDL.

Les systèmes d'exploitation temps réel (SETR) admettent comme entité structurante élémentaire, la tâche. Un ensemble de tâches s'exécutent en concurrence pour réaliser une fonction. A leur tour, un ensemble de fonctions peuvent être rassemblées pour effectuer des fonctions plus complexes, jusqu'à constituer une application complète. Par ailleurs, la grande majorité des SETR fonctionnent sur le mode des machines à états finis.

SDL offre une représentation graphique adaptée à l'approche segmentée et hiérarchique des SETR, ainsi que les moyens pour concevoir et représenter des machines à états finis. A première vue, il semble que SDL s'accorde avec l'architecture des systèmes temps réel mais

la réalité technique n'est pas si évidente, et certaines attentes des développeurs temps réel restent insatisfaites.

Tout d'abord, les types de données abstraits tels que ADT ne correspondent pas au développement de nouvelles solutions. A vouloir proposer un type de données trop ouvert pour éviter d'exclure des données, ADT accuse un manque de précision important qui ne correspond pas à l'utilisation des concepteurs de solutions temps réel. L'optimisation passe souvent par les données et surtout les types de données ; les développeurs se retrouvent frustrés de ne pas retrouver la même précision dans SDL que dans des langages usuels comme le C. Ceci entraîne également des difficultés de correspondance entre les types provenant d'un langage de programmation compilé et les types utilisés dans les spécifications SDL. De plus, il n'y a pas de compilateur générique pour les spécifications SDL, ainsi il faut invariablement passer par une traduction en C avant d'implémenter sur une cible.

Ensuite, certains concepts SDL ne correspondent pas au fonctionnement des SETR. Dans SDL, le concept de priorité s'applique aux messages alors que les SETR l'appliquent aux tâches. Par ailleurs, SDL considère qu'une transition ne peut être interrompue alors que dans un SETR l'exécution peut être interrompue à n'importe quel moment, notamment lorsqu'on admet des priorités entre les taches.

Aussi, il y a des concepts triviaux pour les développeurs d'applications temps réel qui n'existent pas dans SDL, par exemple les sémaphores. Ce moyen banal pour synchroniser les taches et éviter les blocages inter processus n'a pas lieu d'être dans un système réparti.

Pour les raisons évoquées, les outils fondés sur la recommandation SDL ne sont pas vraiment adaptés pour les développements d'applications temps réel et notamment celles accédant à des ressources partagées. Ceci a eu pour effet de faire naître une multitude de solutions alternatives, hybrides permettant aux utilisateurs de SDL d'intégrer du code C dans leurs spécifications, rompant avec le standard de l'UIT. Ce phénomène a été constaté dans un nombre de grandes entreprises telles qu'Alcatel, Nokia, Nortel, EADS ou encore Sagem.

Encore une fois, ce déroulement affaiblit le standard SDL et ne le rend accessible qu'à une élite ayant les moyens financiers et le temps nécessaire pour mettre en place une chaîne logicielle dérivée de la norme pour répondre à leurs besoins spécifiques.

Plus encore, l'arrivée d'UML bouscule le monde des langages de modélisation et SDL entre autres, doit se repositionner en tant que langage de modélisation graphique. La tendance actuelle étant de faire de SDL le complément temps réel d'UML. Mais pour arriver à ce résultat, il faudra envisager une série de modification du langage :

- Ouvrir le langage SDL aux types de données externes. Aujourd'hui il accepte le type ASN.1 issus des télécoms, mais il manque encore une prise en compte des autres types de données provenant du C, C++ ou ADA.

- Modifier les incohérences sémantiques concernant les transitions entre états, la gestion des canaux de communication, etc., pour se rapprocher de la réalité des systèmes temps réel.

- Proposer des extensions à d'autres concepts temps réel tels que les sémaphores, et permettre une gestion plus fine des processus.

2.2.2 Les outils de développement SDL

Une majorité d'entreprises du secteur des télécommunications utilise à ce jour des outils à base de SDL pour concevoir et décrire leurs systèmes. Cependant, les outils d'hier ne sont pas les outils d'aujourd'hui, et les concepteurs de logiciels cherchent à se rapprocher des problématiques de leurs clients et de sentir les tendances du marché. Les axes de développement du langage SDL ont été dictés par les utilisateurs et incorporés ensuite dans les applications produites par les concepteurs de logiciels.

Les évolutions du langage SDL ont été longuement discutées et représentent des orientations théoriques. Les concepteurs sont plus pragmatiques et proposent de plus en plus de solutions hybrides, tentant de s'approprier le meilleur de plusieurs mondes. Mais quelle consistance trouve-t-on dans ces outils associant SDL, UML, C, et autres langages ?

Par ailleurs, la plupart des outils implémentent au mieux la version 96 de SDL et souvent partiellement seulement. Il faut se rendre à l'évidence que les éditeurs suivent difficilement les modifications importantes et fréquentes du standard. Ces derniers se raccrochent tant bien que mal à un standard qui ne leur convient pas entièrement.

De même que pour la description des concepteurs, les paragraphes suivants s'inspirent des présentations officielles des outils par leurs créateurs et de fait ne sont pas impartiales.

2.2.2.1 Cinderella SDL

Cinderella SDL est un produit de la société danoise Cinderella édité pour la première fois en 1998. Cette application permet d'établir des spécifications SDL, de les valider et de les tester via des simulations.

Cinderella SDL 1.3 prend en compte les notations SDL-92, SDL-96 et SDL-2000, mais elle ne précise pas dans quelle mesure elle s'adapte aux différentes sémantiques. Elle reconnaît également les types de données ASN.1 et fait correspondre une visualisation graphique à toute notation textuelle.

Cinderella SDL livre aussi une API permettant de piloter les systèmes conçus avec le logiciel dans le but de permettre le développement d'interfaces personnalisées.

Dans le cadre de ce mémoire, j'ai téléchargé la version d'évaluation de Cinderella SDL dans le but de me faire une idée plus précise d'un logiciel de ce genre. Au premier abord, l'outil offre beaucoup de possibilités. La prise en main n'est pas intuitive pour celui qui n'a jamais fait de SDL et la prise en compte des différentes versions de SDL est assez déroutante. Le logiciel semble essentiellement orienté sur SDL-96, proposant encore l'entité de structure *service*, apparue dans la version SDL-96 et qui a disparu dans SDL-2000 au profit de *l'agrégat d'états*. De manière générale, il semble que les éditeurs ont des difficultés à suivre les évolutions du langage SDL.

Probablement l'outil le plus simple parmi les trois cités ici, sa version la moins coûteuse vaut 2500€ pour une licence poste unique et 5000€ pour une licence réseau.

2.2.2.2 SDL Tau

SDL Tau est une application pour concevoir et implémenter des systèmes complexes communiquant par signaux. Elle permet également d'effectuer des tests automatisés sur le système.

Plus précisément, SDL Tau permet :

- La spécification détaillée d'un système,
- La simulation et la vérification de son comportement,
- La génération de code en C/C++ à partir d'un modèle validé,
- La définition, la génération, et l'exécution de tests sur un modèle.

La version de SDL employée dans cet outil n'est mentionnée nulle part, ce qui laisse présupposée qu'il n'a pas subit d'évolution pour intégrer pleinement SDL-2000.

En effet, SDL Tau n'est pas l'outil phare de la société Telelogic, contrairement à son équivalent UML. Ce dernier intègre déjà les concepts UML 2.0 standardisés au courant de cette année. Certains pourront y voir un signe avant coureur de l'abondant de SDL par l'outilleur.

2.2.2.3 RDTS G3

« Real Time Developer Studio G3 » (RTDS G3) est un outil multi plateforme d'édition et de simulation pour SDL comprenant les modèles SDL-88, SDL-92 et SDL-96, mais pas SDL-2000. Cet outil est intéressant car il propose une vision légèrement différente des outils précédents en se plaçant dans une optique de développement temps réel sur des systèmes d'exploitation temps réel (SETR).

RTDS G3 propose principalement :

- Un outil de simulation et de test de scénarios,
- Une vision graphique mêlant SDL et UML des systèmes conçus,
- L'intégration de bibliothèques de code en C, C++, Python, etc.,
- La génération code à partir des spécifications,
- La possibilité d'établir des diagrammes MSC par la trace des simulations effectuées ou par des scénarii pré-établies,
- La gestion en parallèle de SDL, UML et SDL-RT.

RTDS G3 permet des diagrammes de classe et des diagrammes de déploiement issus du langage UML, mais utilisant aussi des symboles spécialisés de SDL. L'outil joue ainsi sur la complémentarité d'UML et SDL : UML pour définir l'aspect statique du système et SDL pour l'aspect dynamique.

SDL-RT est une extension de SDL incorporant des notions de langages de programmation temps réels comme le C ou le C++ plus proches de l'implémentation finale. Prise en compte des sémaphores et les pointeurs. Il permet également un déboguage graphique sur la base des débogueurs C/C++.

Cette optique est intéressante car elle offre enfin un pont entre le langage SDL et le développement temps réel sans restriction apparente. Plus qu'une offre commerciale, les

caractéristiques de SDL-RT sont disponibles sur Internet et proposés comme un standard possible par ses créateurs.

Mais est ce que le pragmatisme de SDL-RT peut favoriser un regain d'intérêt pour une forme de SDL ? Ou au contraire, est-il trop proche de l'implémentation finale pour s'adapter aux évolutions futures du marché ?

Ces questions stigmatisent la problématique actuelle du langage SDL, trop spécifique pour certains et pas assez généraliste pour d'autres.

Chapitre 3 - Convergences et divergences entre SDL et UML

Le langage SDL est apparu une décennie avant la généralisation de la modélisation objet. Celle-ci a réellement percée au cours des années quatre vingt, provoquant une déferlante de langages et de méthodes pour concevoir et spécifier des systèmes d'information en objet.

A la fin des années quatre vingt dix, est apparu le « Langage de Modélisation Unifié » (UML), compromis entre plusieurs langages existants alors. Rapidement confirmé comme le standard en terme de modélisation objet, nous le retrouvons quelques années plus tard dans l'enseignement des méthodes d'analyse et de conception de systèmes d'information.

Aujourd'hui, UML se généralise dans la conception de systèmes d'information et de plus en plus de d'utilisateurs cherchent à canaliser la flexibilité et l'expressivité de ce langage pour répondre à des besoins spécifiques. De fait, il existe aujourd'hui une grande variété de domaines utilisant UML, et notamment celui du développement temps réel.

Il s'avère que les langages SDL et UML sont employés sur des projets similaires en substance. En partant de ce constat, il est légitime de se demander quels sont leurs points communs et leurs différences ?

Pour mettre en évidence les points communs et les différences des deux langages nous allons d'abord présenter succinctement UML. Ensuite, sur la base des travaux du projet CONVERGENCE, nous allons établir les forces et faiblesses des deux langages. Et finalement nous présenterons une lecture de l'évolution commune de ces langages : le métissage de SDL et UML comme solution unifiée.

Ce chapitre étudiera essentiellement les langages UML 1.3 et SDL-96 car ce sont les versions les plus répandues et les mieux intégrés actuellement. Nous évoquerons brièvement les évolutions présentes dans les dernières versions, UML 2.0 et SDL-2000, bien que leur impact soit difficilement qualifiable aujourd'hui.

3.1 Les concepts d'UML

Le « Langage de Modélisation Unifié » (UML/LMU, « Unified Modeling Language ») est un standard de modélisation objet recommandé par l'OMG (Object Management Group) depuis 1997. Il représente un effort collectif pour édifier une norme au milieu d'une pléthore de solutions isolées. Les principaux auteurs de cette norme sont James Rumbaugh, Grady Booch et Ivar Jacobson.

Le langage UML propose aux concepteurs un moyen unique et consistant pour analyser et concevoir des systèmes à partir de concepts objets, en permettant de spécifier, visualiser, construire et documenter la structure et le comportement d'un système.

En 1997, UML présentait peu d'évolutions par rapports aux langages de modélisation existants. Toutefois, la facilité d'apprentissage et d'utilisation de ce langage, ainsi que ses graphiques très explicites ont largement contribué au succès de cette norme.

3.1.1 Architecture d'UML

L'architecture d'UML se décompose en 4 niveaux, et représente une architecture de méta-modèle hiérarchique. Le niveau 0 contient les objets à l'exécution, le niveau 1 est le niveau modèle contenant les classes, le niveau 2 correspond au méta-modèle UML et le niveau 3 et celui du MOF (Meta Object Facility, langage propre à l'OMG pour définir les méta-objets et les méta-data).

La relation entre deux niveaux et de « type-instance », les éléments d'une couche inférieure, sont des instances des éléments de la couche supérieure.

Cette organisation est importante car elle explique en grande partie la flexibilité et l'attrait du langage UML pour la représentation de systèmes très divers.

3.1.2 Les Classes

Une classe représente une abstraction employée dans la modélisation de systèmes. Les classes représentent le concept central dans l'orienté-objet et elles occupent une place similaire dans UML.

Une classe décrit les propriétés partagées par une collection d'objets. Ces propriétés peuvent être de sorte attribut, opération, association, relation (avec d'autres classes), ou encore spécification de comportement.

Elles peuvent être actives ou passives en fonction des propriétés de concurrence affectées à l'instance d'une classe.

Les classes pouvant être instanciées sont appelées concrètes, et celles qui ne le peuvent pas sont des classes abstraites (virtuelle). Ces dernières vont servir la structure hiérarchique des classes et devront être redéfinies dans une sous-classe afin d'être instanciées en tant qu'objets.

Une classe possède une structure et un comportement. La structure comprend des attributs, et le comportement est constitué d'opérations. Les éléments de la classe ont des propriétés qui définissent leur visibilité.

Un aspect des opérations d'une classe qui nous intéresse tout particulièrement est la concurrence des opérations. Ceci nous explique comment les opérations se comportent lorsqu'elles existent de manière concurrente avec d'autres opérations : Elles peuvent être séquentielles ou concurrentes.

Une opération UML est implémentée par une ou plusieurs méthodes. Une méthode précise l'implémentation d'une opération. Ceci permet le polymorphisme des méthodes et la séparation entre la spécification et l'implémentation.

Les classes peuvent être associés à d'autres classes, ou hériter d'une autre classe.

Les associations sont définies entre deux classes et donnent des informations sur les rapports qui unissent les classes.
L'héritage entre classes donne à une classe des propriétés d'une classe parente. Celle-ci dispose alors de propriétés locales en plus des propriétés héritées des parents. Les opérations peuvent être redéfinies par les héritiers. Ainsi une nouvelle méthode peut être affectée à une opération héritée, mais elle ne peut pas être supprimée par héritage.

L'interface est un moyen de contrôler l'accès à une classe. Elle offre un moyen de décrire l'accès à une classe en présentant des opérations disponibles pour accéder à une classe. Ainsi, les entités accédant à l'interface n'ont pas d'information explicite sur la structure de la classe sous-jacente.

3.1.3 Les diagrammes UML

Les diagrammes UML sont grandement responsables de la popularité du langage. Leur richesse et leur variété facilitent la compréhension et la communication à partir d'un modèle.

Il existe neuf diagrammes UML, proposant chacun une vision différente, même si certains partagent des informations. Chaque diagramme montre un aspect spécifique du système existant ou à venir.

3.1.3.1 Diagramme de Classes

Le diagramme de classes contient une collection de classes, de relations, d'associations entre classes et d'interfaces. Il modélise une vue statique du système et donne une base pour les diagrammes suivants. Nous retrouvons des traces du diagramme de classes dans pratiquement tous les diagrammes suivants.

La description des types en SDL est proche des diagrammes de classes dans le principe.

3.1.3.2 Diagramme d'Objets

Cas particulier du diagramme de classe, il ne contient que des instances des classes pré définies. A fortiori nous pouvons retrouver plusieurs instances de la même classe. Il montre une vision sensiblement différente des classes au moment de l'exécution.

3.1.3.3 Diagramme de Cas d'Utilisation

Diagramme informel, décrivant ce que fait un système de manière générale et à des niveaux de détails très variés. Ce diagramme n'explique pas précisément le comportement découlant des cas d'utilisation. Son utilisation principale est la présentation clair des fonctionnalités d'un système comme base de discussion durant les phases d'analyse et de conception. Il peut être une aide précieuse pour valider le fonctionnement d'une application par l'utilisateur final.

3.1.3.4 Diagramme d'Etats

Le diagramme d'états est le modèle le plus courant pour spécifier le comportement d'un système. Il se fonde sur une représentation graphique d'automates finis. Plus précisément, ce

sont des diagrammes d'états et de transitions décrivant le comportement de machines à états lors de la réception de stimuli.
La notation graphique du comportement dans SDL est proche des diagrammes d'états. Ils se fondent tout deux sur les automates finis.

3.1.3.5 Diagramme d'Activité

Le diagramme d'activité symbolise un graphe d'activité, un type particulier de machine à états finis. UML propose cette variation dans la notation des machines à états pour préciser certains aspects du flux de données.

3.1.3.6 Diagramme de Séquences

L'interaction est représentée par les diagrammes de séquences et les diagrammes de collaboration. Nous reproduisons ainsi d'une manière descriptive l'aspect dynamique d'un système, mais ce n'est pas un moyen de décrire le comportement. Il manque la possibilité de spécifier le déclenchement d'un comportement.

Le diagramme de séquence met l'accent sur l'échange de messages entre objets au cours du temps. Il présente les messages dans un ordre chronologique et montre clairement le flux des messages et les événements au cours du temps. Il montre clairement la durée de vie des objets et donne une visualisation graphique du statut d'un objet (effectue une action, attend une réponse, etc.).

C'est un parent proche du « Message Sequence Charts » (MSC), recommandation Z.120 de l'UIT. Toute fois, le MSC comprend quelques propriétés supplémentaires.

3.1.3.7 Diagramme de Collaboration

Ce dernier a un autre type de représentation des interactions proposant des objets et les relations existantes entre les rôles des objets mais offre une vision limitée de l'aspect temporel. Il se focalise essentiellement sur l'organisation des objets participants à une interaction.

3.1.3.8 Diagrammes de Composants

Le diagramme de composants montre les dépendances entre éléments logiciel, code source, code compilé, librairie, tables, documents, etc.

3.1.3.9 Diagramme de Déploiement

Le diagramme de déploiement est la partie exécutable des deux, il permet de voir l'organisation actuelle ou future des composants. Permet d'établir une stratégie de déploiement et de contrôle des versions.

3.1.4 La Structure

La structuration du système est définie par le *paquetage*, le *sous-système* et le *modèle*.

Le moyen de groupement le plus utilisé est le *paquetage*, permettant de regrouper les spécifications UML dans un seul lot nommé. Il représente un espace de noms commun pour tous les éléments qu'il contient. Le concept de *paquetage* n'est pas sans rappeler les librairies en programmation. Un *paquetage* peut contenir d'autres *paquetages*.

Un *modèle* est une sorte de *paquetage* contenant la description entière d'un système modélisé.

Un *sous-système* est un type de *paquetage* augmenté de propriétés additionnelles et auquel on peut indiquer le comportement de ses composants. Dans la pratique, il s'avère cependant très difficile à employer par manque de définition sémantique.

Ces éléments de structures permettent de segmenter l'univers modélisé en unités logiques afin de clarifier la représentation du système et de mettre en évidence les rapports entre les entités. Cette volonté de structurer le système ressemble beaucoup à la division en *système*, *bloc* et *processus* des systèmes vus dans SDL. Mais les concepts sont beaucoup moins typés dans UML, témoignant d'une vision plus généraliste du langage.

3.1.5 Le Comportement

Le comportement en UML est décrit au *niveau système*, *classe* ou *opération* :

- Au *niveau système*, le comportement est décrit au stade initial de l'analyse et de la conception, il n'est que descriptif. Pour ce faire, UML propose les cas d'utilisation et les diagrammes d'interaction, ainsi que les machines à états et les activités.

- Au *niveau classe*, le comportement est décrit en utilisant exclusivement les machines à états et les activités, les deux donnant une perspective du comportement.

- Au *niveau opération*, il est possible de représenter le comportement par une opération des machines à états ou les actions.

Les machines à états finis d'UML sont des variantes des machines à états de Harel, adaptés aux caractéristiques de l'orienté-objet. Le standard UML décrit les machines à états mais ne donne pas de tous les détails quant à la sémantique dynamique de ces machines à états.

Dans des cas bien définis, les transitions UML entraînent le changement l'état d'un système : déclenchement d'un événement, réception d'un message, activation d'une condition. Les machines à états UML peuvent avoir des actions attachées aussi bien aux transitions qu'aux états.

Les transitions sont déclenchées lors d'un événement et font passer le système d'un état dans un autre, éventuellement le même. Des conditions de garde existent et peuvent prévenir une transition. Quand une transition se déclenche, on sort de l'état source, on effectue une transition et on entre dans un état de destination.

UML définit une collection de règles et de conditions telles que les priorités entre transitions.

Ainsi les machines à états présentés par UML peuvent être d'une complexité importante et offrent aux concepteurs les moyens de décrire des systèmes eux-mêmes très complexes.

Néanmoins certaines lacunes de la sémantique dynamique du langage empêchent une interprétation non ambiguë du comportement, contrairement à SDL.

3.1.6 La Communication

La communication dans UML passe par le biais de l'adressage direct ou le « broadcast » (diffusion) : Un objet peut diffuser un message à un ensemble d'entités, ou il peut l'envoyer à un autre objet via l'interface de ce dernier.

De plus, cette communication peut être unidirectionnelle ou bidirectionnelle : La communication bidirectionnelle est accomplie en appelant des procédures, alors que la communication unidirectionnelle se fait par envoie de signaux.

Dans UML, la communication est fondée sur les événements. Les instances d'événements sont générées en réaction à une action à l'intérieur du système ou provenant de l'environnement du système. Et un événement est distribué à une ou plusieurs cibles.

Les stimuli asynchrones correspondant aux événements sont envoyés à un objet et sont stockés dans sa pile d'entrée en attendant d'être traiter par son automate fini.

Par ailleurs, UML définit plusieurs types d'événements :

- Appel (Call), est déclenché quand une opération d'invocation est reçue.
- Signal (Signal), se déclenche à la réception d'un signal.
- Changement (Change), arrive lorsqu'une condition booléenne est satisfaite.
- Temps (Time), se déclenche à l'échéance d'un chronomètre.

3.1.7 Les Données

Il n'existe aucun support de types de données complexes dans la norme UML. Cependant, les outils actuels proposent des correspondances pertinentes avec la plupart des langages objets.

3.2 Forces et faiblesses des langages SDL et UML

A la lueur des concepts d'UML il semble clair que les deux langages partagent un certain nombre d'éléments : l'aspect graphique, une structure orienté-objet, une communication par signaux et procédures distantes, ou encore la représentation des comportements d'un système par des automates finis. Cependant, des nuances importantes distinguent ces langages et forment les forces et les faiblesses de chacun.

Ici, nous allons tenter de préciser le rapport existant entre ses deux langages, en éclairant quelques points forts et lacunes de SDL-96 et UML 1.3. Par ce biais, nous voulons mettre en évidence une complémentarité possible.

3.2.1 Forces & faiblesses de SDL-96

Le langage SDL a déclenché une vague d'enthousiasme chez les concepteurs de systèmes complexes temps réels, avides de disposer d'un outil graphique structurant sur la base d'un langage formel.

Dans un milieu où les outils graphiques sont rares, les diagrammes SDL représentent un apport non négligeable dans la structuration et la compréhension des systèmes complexes. Non seulement un moyen de communiquer autour d'un projet, c'est un véritable outil de conception accompli.

De plus, avec la maturité des outils, les utilisateurs peuvent bénéficier de la nature formelle du langage SDL. La syntaxe stricte et la sémantique bien définie de SDL permettent d'implémenter des outils pouvant vérifier le système modélisé, effectuer des simulations, générer des tests de couverture pour les automates, exporter le système vers un langage de programmation compilé, et finalement tester le code généré.

Malheureusement, le langage SDL est fortement limité à la description des comportements de systèmes réactifs. Encore largement ancré dans le domaine des télécommunications, SDL permet principalement de spécifier des systèmes complexes, temps réels, répartis, communiquant par signaux. Or ce type de système correspond essentiellement aux composants des réseaux télécoms et ne se prête pas à la modélisation précise d'une majorité des systèmes embarqués.

D'ailleurs, même dans les télécoms la spécification SDL n'est pas toujours suffisante dans la phase de conception, et les entreprises choisissent l'utilisation d'outils alliant certains concepts SDL pour décrire le système avec des langages de programmation servant à l'implémentation finale. Ceci leur permet également de se défaire du type de donnée abstrait de SDL, trop imprécis, au profit des types complexes des langages utilisés pour l'implémentation sur cible.

3.2.2 Forces & faiblesses de UML 1.3

Depuis son officialisation en 1997, UML n'a cessé d'étendre son domaine d'action. Résultat de plus d'une décennie des méthodes de modélisation orienté-objet, il offre les moyens pour décrire sous plusieurs angles la structure statique et dynamique de tous types de système.

Son architecture à plusieurs niveaux et sa parfaite intégration des concepts objets rendent le langage UML très flexible. Le langage UML permet de définir des objets, leurs comportements et leurs associations avec d'autres objets. A cela s'ajoute la facilité de réutilisation et d'extension des entités définies. Il devient évident qu'UML peut trouver des applications dans des domaines très variés.

Le grand nombre de types de diagrammes disponibles dans UML permet de souligner certains aspects des systèmes décrits. Il est plus facile de discuter des fonctionnalités d'une application avec un utilisateur non informaticien devant un cas d'utilisation qu'au travers d'un diagramme de classes. Par ailleurs, le diagramme de classe permet d'identifier toutes les entités du système et de les lier facilement alors que le diagramme d'états offre une visibilité réduite à une opération. Finalement, tous ces angles d'approche permettent à l'utilisateur de choisir la façon de décrire le système en offrant une visibilité global ou partielle, exhaustive ou particulière, statique ou dynamique. Cette grande richesse de représentation ouvre le langage aux utilisateurs de tous secteurs au moins pour une modélisation sommaire.

A ses débuts, UML a déclenché un effet de mode qui lui a valu un développement fulgurant dans les départements informatique. Mais les chercheurs ont rapidement vu les faiblesses des sémantiques UML l'empêchant de supporter efficacement des outils plus pointus. La pratique

a prouvé que les sémantiques formelles servent la réalisation d'outils permettant de vérifier, simuler, générer du code ou encore d'améliorer le contrôle de compatibilité entre deux versions d'un système. Toute fois, l'existence d'une sémantique informelle ne prévient pas la réalisation de ce but, il encourage simplement la création de formalismes divergents. Dans UML 1.3, cette faiblesse se caractérise par les omissions et ambiguïtés du langage, les situations non prévues et certaines limitations du langage.

3.2.3 La complémentarité

Les langages SDL et UML, bien qu'issus de deux réflexions différentes, présentent de similarités indéniables. D'une part, les deux langages fondent leur représentation des comportements d'un système sur les automates finis. Et d'autre part, ils présentent une modélisation des systèmes inspirée de l'orienté-objet, incorporant des notions de type et d'instance de type, d'héritage entre les entités, ou encore de redéfinition d'éléments.

Par ailleurs, leurs divergences se fondent principalement sur deux politiques différentes. UML est issu de la réflexion sur l'orienté-objet et se veut capable de représenter n'importe quel système. Alors que SDL provient d'une collaboration d'entreprises et de chercheurs dans le domaine des télécommunications pour créer un langage de spécification et de description.

Ainsi, SDL montre toute la rigueur d'un langage formel, mais ne s'applique correctement qu'aux systèmes rencontrés dans les télécommunication ou systèmes embarqués simples. Quand UML permet une application beaucoup plus large, mais ne dispose pas des moyens sémantiques pour spécifier précisément un système et notamment son comportement.

De ce constat a émerger une volonté de faire converger les deux langages afin que chacun profite des aspects positifs de l'autre.

3.3 Vers un métissage des langages

Le projet CONVERGENCE, lancé par le RNRT vers 1999 avait pour volonté de faire le point sur les langages SDL et UML, et proposer des évolutions pour les deux langages.

La majorité des propositions soumises à l'OMG et à l'UIT ont trouvé leur chemin dans les versions SDL-2000 et UML 2.0. Ainsi, nous avons vu apparaître, entres autres, une nouvelle structure hiérarchique par agents dans SDL-2000, la création d'un concept d'interface directement inspiré d'UML, ou encore une spécialisation des diagrammes d'états d'UML avec des concepts provenant de SDL. Ces changements semblent prouver la pertinence des points soulevés dans ces langages en rapport avec la modélisation de systèmes embarqués temps. Mais ils ne sont qu'un premier pas dans la direction du métissage des deux langages.

Le développement et l'évolution d'un langage sont des tâches ardues. A fortiori, pour deux langages, c'est encore plus difficile.

Afin de prévenir les dérives, l'équipe du projet CONVERGENCE, préconise de conserver la distinction entre SDL et UML, tout en faisant évoluer les concepts de chacun en jouant sur la complémentarité des deux langages. Pour ce faire, il faut effectuer un rapprochement des notations des langages pour faciliter une correspondance syntaxique. D'autre part, il faut préserver les concepts spécifiques de SDL ayants déjà fait leurs preuves sur le terrain. Cela dit, il est aussi nécessaire de combler les faiblesses de chacun en s'inspirant de l'autre,

notamment l'aspect objet de SDL. Mais surtout il est indispensable de garantir la compatibilité entre SDL et UML, en modélisant les concepts SDL dans UML et en proposant des outils sémantiques (sur la base des sémantiques SDL) pour les éditeurs UML.

Par ce biais, UML deviendra un langage plus solide, et notamment sur le terrain du temps réel. Il permettra de proposer une réelle alternative pour les télécommunications, prenant en compte les concepts spécifiques aux architectures complexes, et le potentiel des langages formels. Cela facilitera également la transition d'outils génériques UML vers des outils dédiés à la conception de systèmes spécifiques, par exemple du temps réel. Et pour les utilisateurs de SDL, leur investissement actuel dans les modèles et les compétences SDL sera préservé.

Nous pouvons affirmer que le métissage est en marche. Seulement, il faudra plusieurs années pour permettre aux outilleurs d'intégrer pleinement les évolutions. Et c'est à ce moment seulement que nous aurons le fin mot de l'adéquation du langage UML, de la structure et sémantique SDL, et des systèmes embarqués temps réel.

Conclusion

SDL est un langage formel orienté-objet permettant de décrire et de spécifier des systèmes informatiques complexes, réactifs et communiquant. Ces propriétés font de SDL un langage idéal pour concevoir, décrire, valider et simuler des systèmes temps réel, distribués, communiquant par signaux, tels que les systèmes trouver en télécommunication.

Répondant directement à une demande du marché, ce standard a connu un succès rapide et les premiers outils ont fait l'unanimité dans le secteur des télécommunications. Les propriétés du langage SDL ont fait de lui un acteur clé de la conception de systèmes télécoms et il a rapidement été pré senti comme un moyen de représenter tous les systèmes complexes temps réels.

Malheureusement, il semble que SDL a atteint ses limites. Les spécialistes sont mitigés sur l'utilité du langage, certains le trouvant trop proche des systèmes télécoms pour en faire une utilisation généralisée. Effectivement, les entreprises en télécommunication, avides d'optimiser leur chaîne de production logicielle, ont dores et déjà adoptées des solutions plus proches de l'implémentation finale. Ces solutions sont développées par les éditeurs de logiciels qui ont réagi plus rapidement que les organismes de normalisation et proposent désormais des solutions hybrides à base de SDL et UML.

Le SDL apparaît alors comme un langage de spécialistes essentiellement employé par une niche de sociétés en télécommunication. Et même cette niche est en train de le délaisser au profit de solutions alternatives et notamment UML.

Bousculé par l'arrivée du langage de modélisation objet UML, SDL a du se repositionner sur le marché des outils de modélisation graphique. Fort d'une certaine maturité, le langage SDL semble encore tenir tête sur la niche des télécoms, mais UML gagne tous les domaines nécessitant des outils de spécification et de description des systèmes d'information.

UML séduit par sa flexibilité ainsi que par sa capacité à s'adapter aux cas les plus spécifiques, comme aux cas généraux. Cependant, les utilisateurs avertis cherchent plus qu'un langage de description et rapidement UML a montré les limites de sa sémantique.

Proche du langage UML par son orientation objet et sa description du comportement des systèmes via les automates finis, SDL a été montré en exemple pour la spécification de systèmes embarqués communiquant. Rapidement, il est apparu que les deux langages pouvaient se compléter.

Les premiers efforts de rapprochement entre SDL et UML prennent forme dés 1999, afin de réunir le meilleur de deux mondes : Le modèle souple et évolutif d'UML allié à la rigueur de la sémantique de SDL pour spécifier les systèmes embarqués communiquant. De ce consensus, UML semble être le grand gagnant. Son modèle à plusieurs niveaux, plus modulaire que celui de SDL, son intégration parfaite des concepts objets et ses diagrammes variés font d'UML le candidat idéal pour une méthode de modélisation unique, quelque soit le domaine.

Cependant, il est illusoire de prétendre pouvoir atteindre un niveau de détail suffisant pour décrire avec précision tout types de systèmes. Ainsi une spécialisation des modèles est nécessaire pour permettre de valider, tester, générer une solution. SDL sera à la base de cette spécialisation pour le temps réel !

Publié cette année, UML 2.0 semble se diriger dans ce sens, prenant en compte les remarques de travaux tels ceux réalisés dans le cadre du projet CONVERGENCE et proposant des moyens d'étendre le langage UML par des profiles spécialisés.

Clairement, SDL est un langage de spécification d'une grande rigueur adapté pour la spécification détaillée d'un système de télécommunication. Cependant il pêche pour décrire d'autres systèmes complexes et il s'avère trop compliqué pour des modélisations simples.

Les évolutions constantes du langage ont permis de prendre en compte les besoins des utilisateurs, mais le rythme des mises à jours semble en désaccord avec les concepteurs de logiciels qui sont obligés de sortir de nouvelles versions en permanence. A tel point que ces derniers proposent des versions propriétaires intégrant une partie des notions de SDL, et parfois même en l'associant à des langages de programmation utilisés habituellement dans le cadre du temps réel.

Ainsi la volonté d'ouvrir et d'enrichir le langage SDL dans la version SDL-2000 semble totalement dénuée d'effets sur l'industrie, malgré cinq années de gestation. La tendance actuelle présente exclusivement UML comme solution de modélisation d'avenir, à tort ou à raison.

En fin de compte, le problème reste le même : Il s'agit de trouver un langage suffisamment ouvert pour tout modéliser, de l'application de gestion, aux réseaux de micro-composants communiquant par signaux, en passant par les contrôleurs temps réels. Les langages objets permettent de modéliser un système complexe mais le détail et surtout la création d'un exemple exécutable nécessite une correspondance fine avec l'univers cible. Nous pouvons nous poser la question du bien fondé d'une convergence de deux langages complémentaires, chacun risquant de perdre son identité en s'adaptant à l'autre.

Personnellement, je pencherai pour la séparation de la modélisation et de l'implémentation. Il me semblerait plus prudent de conserver un certain recul lors de la modélisation pour ensuite implémenter avec précision le système final et éviter les effets de vases communiquant.

L'idéal serait d'avoir un langage possédant plusieurs niveaux d'abstraction indépendants et de pouvoir passer d'un niveau à l'autre pour décrire, spécifier puis implémenter. UML s'oriente vers ce type de modélisation en proposant des mécanismes de spécialisation du modèle. Il reste à voir quelle sera son implémentation dans les outils à venir.

Annexe 1 – Bibliographie

[ART 99] Rodolphe Arthaud,
Verilog,
« UML & SDL, Comparison and Combination »
Projet CONVERGENCE, Janvier 1999.

[BELL 96] Mark A. Ardis, John A. Chaves, Lalita Jategaonkar Jagadeesan, Peter Mataga, Carlos Puchol, Mark G. Staskauskas, James Von Olnhausen.
Bell Laboratories
« A Framework for Evaluating Specification Methods for Reactive Systems »,
IEEE Transactions on Software Engineering, Juin 1996.

[CIND] Cinderella
http://www.cinderella.dk
Brochure Cinderella SDL

[GAU 2003] Emmanuel Gaudin,
PragmaDev,
« SDL to design real-time software »,
SDL Forum Proposal, Juin 2003.

[IEC 2000] International Engineering Consortium,
« Specification and Description Language »
Web ProForum Tutorials, 2000.

[KARL 94] Klaus Lewerentz, Thomas Lindner,
Université de Karlsruhe
« Case Study : Production Cell »
FZI-Publication, Janvier 1994.

[LEBL 99] Philippe Leblanc,
Verilog,
« A stronger UML for Real-Time Development »,
Embedded Systems Conference, Septembre 1999.

[OBER 2001] Ileana Ober,
Telelogic,
« Harmonisation des langages de modélisation avec des extensions orientées-objet et une sémantique exécutable »
Thèse doctorante, Avril 2001.

[RNRT] Réseau National de Recherche en Télécommunications
http://www.telecom.gouv.fr/rnrt

[RTDS G3] PragmaDev
http://www.pragmadev.com
Brochure RTDS G3

[SDL 2000] Union Internationale des Télécommunications – Télécommunication Section Normalisation, Recommendation Z.100:
« Specification and Description Language »,
Electronic Bookshop, Geneva, 2000.

[TAU] Telelogic
http://www.telelogic.com
Brochure SDL Tau

[TIMe 99] TIMe
« SDL Tutorial »
TIMe Electronic Textbook, 1999.

[WIKI 2005] Wikipédia, Encyclopédie libre sur Internet
http://fr.wikipedia.org/

Annexe 2 – Glossaire

ADT : « Abstract Data Type »

ASN.1 : « Abstract Syntax Number One »

CCITT : « Comité Consultatif International Télégraphique et Téléphonique »

ITU / UIT : « International Telecommunications Union » / « Union Internationale des Télécommunications »

MSC : « Message Sequence Chart »

OMG : « Object Management Group »

SDL / LDS : « Specification and Description Language » / « Langage de Description et de Spécification »

RFI : « Request For Information »

RFP : « Request For Proposals »

RNRT : « Réseau National de Recherche en Télécommunications »

TTCN : « Tree and Tabular Combined Notation »

UML : « Unified Modeling Language »

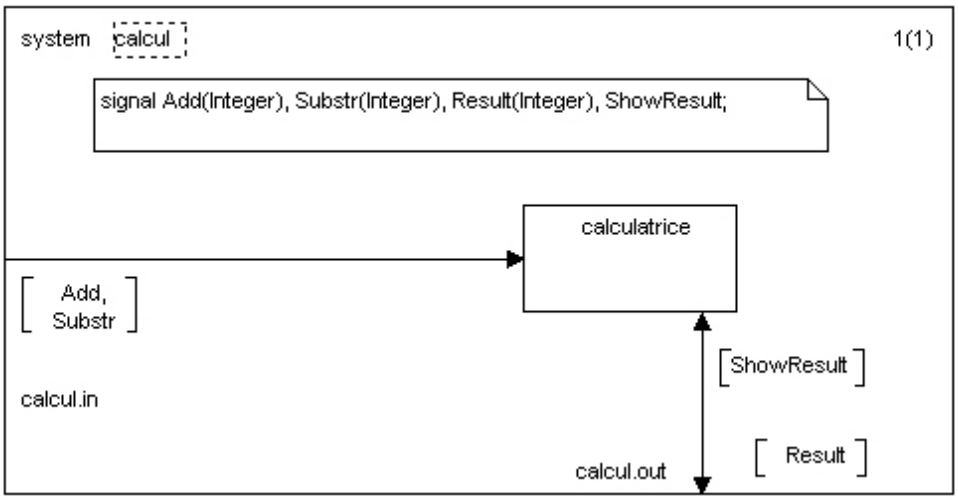
Annexe 3 – Modélisation d'une Calculatrice

Voici la modélisation d'une calculatrice – simplifiée – réalisée à l'aide de Cinderella SDL.

Notre calculatrice permet d'additionner et de soustraire des nombres entiers et renvoie le résultat à la demande.

La description SDL se présente comme suit :

Le système « calcul »

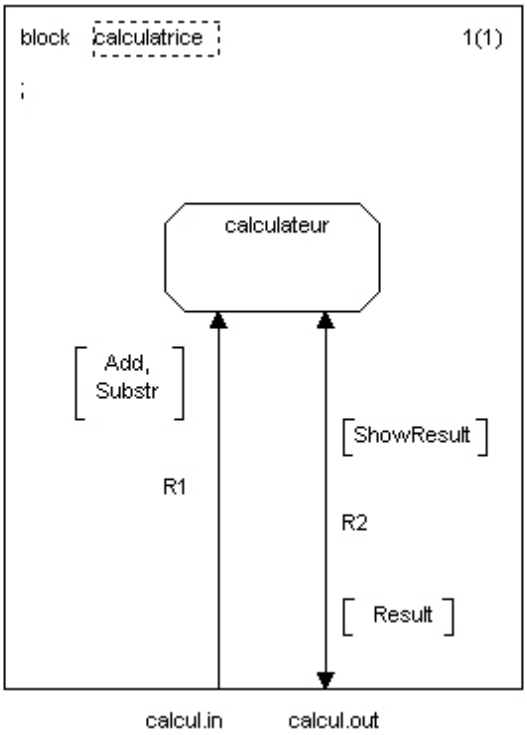


Le système « calcul » comprend un *bloc* (calculatrice) et deux canaux (calcul.in et calcul.out).

Le canal calcul.in est unidirectionnel et ne permet que l'envoie d'une opération d'addition (Add) ou de soustraction (Substr) avec en paramètre la valeur à ajouter ou soustraire du total courant.

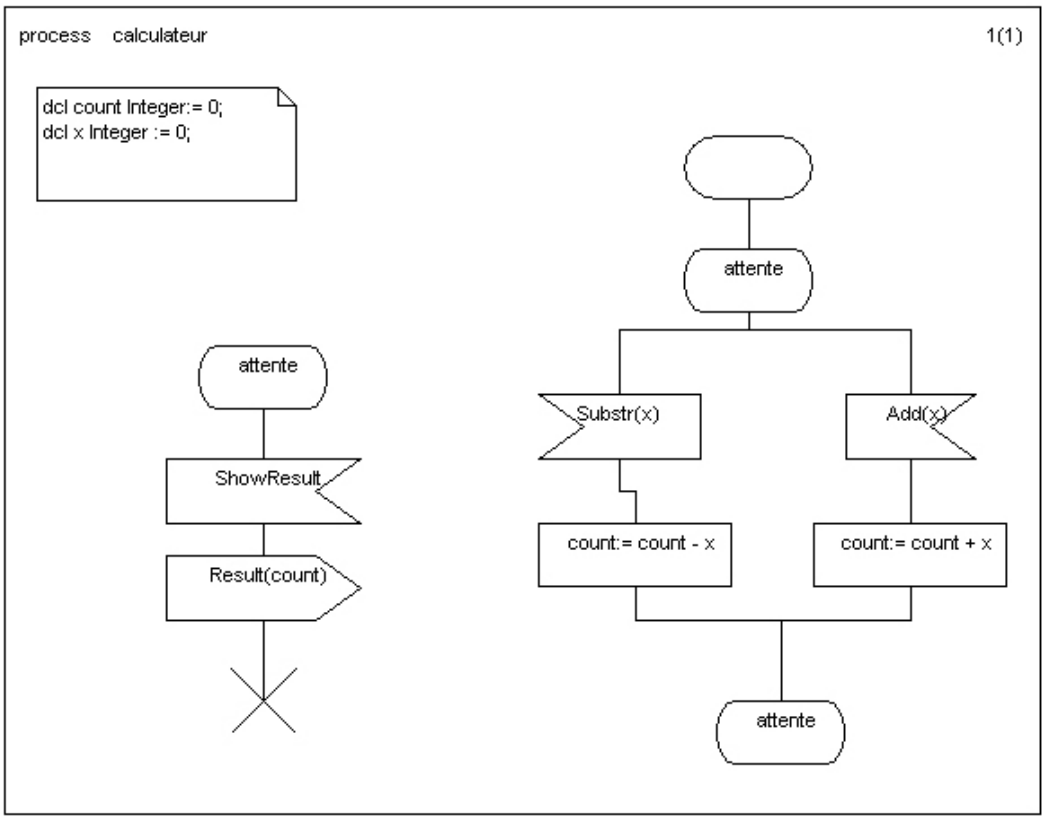
Le canal caclul.out est bidirectionnel, il permet de retourner le résultat courant de la calculatrice à une demande explicite provenant de l'environnement.

Le bloc « calculatrice »



Le *bloc* « calculatrice » relaye les signaux provenant de l'environnement jusqu'au *processus* « calculateur ».

Le processus « calculateur »



Le *processus* « calculateur » est défini par un automate fini. Ce dernier se lance au démarrage du système et se met immédiatement en « attente » d'un signal entrant.

Les signaux « Add » et « Substr », ajoute et retranche respectivement une certaine valeur (x) à la variable « count ».

A la réception du signal « ShowResult », l'automate renvoie la valeur actuelle de « count » et s'arrête.

Une Simulation d'addition

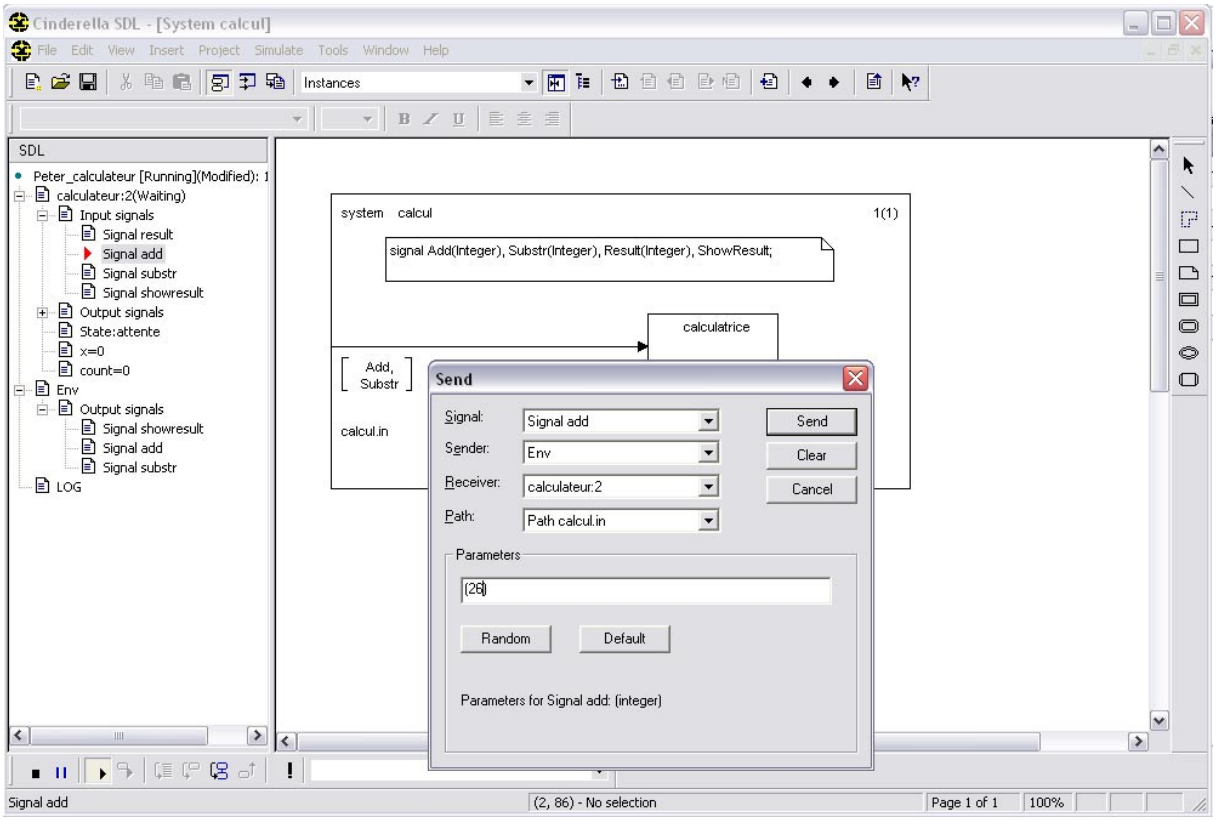


Figure 3 Ajout de la valeur 26 au compte initial (0)

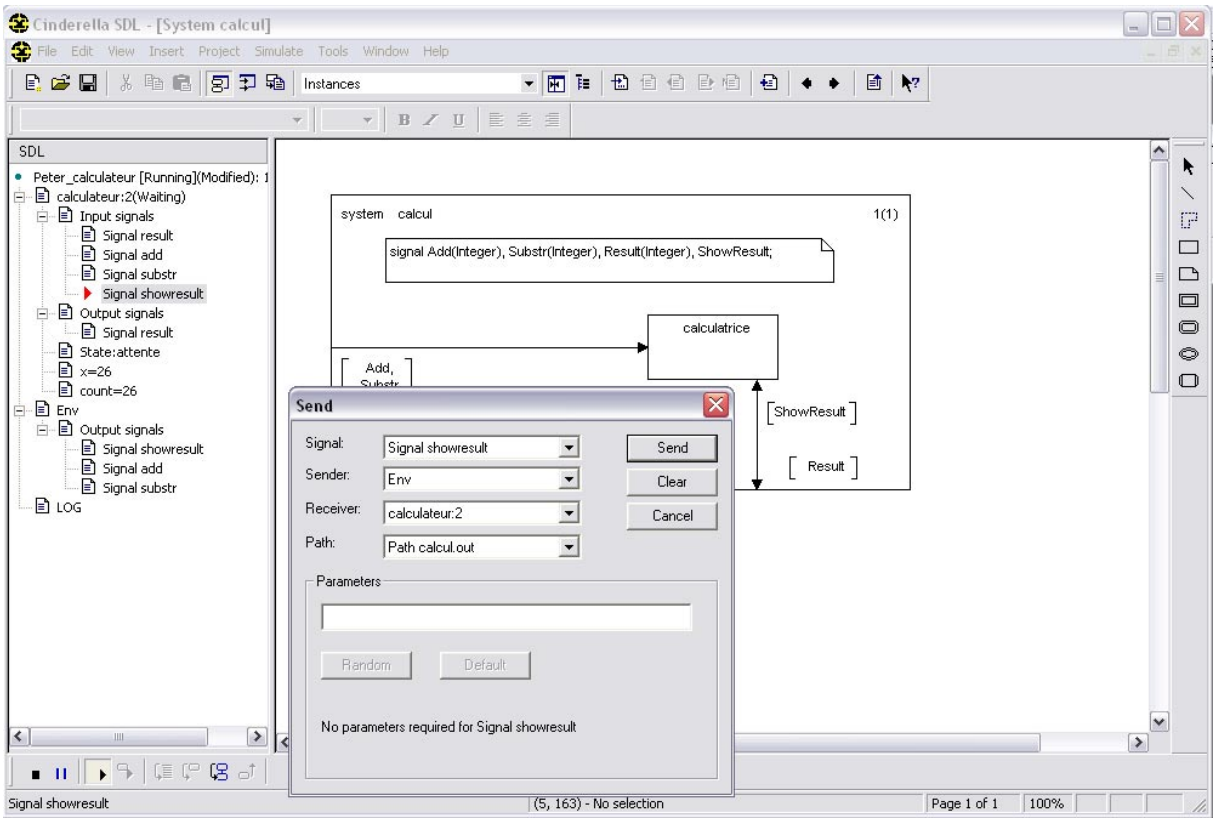


Figure 4 Demande du résultat