
A stronger UML for Real-Time Development

EMBEDDED SYSTEMS CONFERENCE, NOV-1999, SAN JOSE, CA.

Class# 469

Philippe Leblanc

Email: leblanc@verilog.fr
VERILOG

150, r. Vauquelin, BP 1310
31081 Toulouse Cedex, France
<http://www.verilogusa.com>

Abstract

While capturing a large mind-share, the use of UML in the real-time embedded industry is still in infancy. Reasons for the current lack of obvious successes that could propel UML as *the* notation of choice in the real-time industry are multiple. Although UML is the fruit of the blend of solid concepts and technologies, it still lacks some important features which only formal languages have been able to cover so far. Adding some of the key core characteristics of a formal language to UML will break the current limitation that UML is not executable and that no pure simulator or full-code generator can be built-upon, thus removing the barriers to its adoption in certain technical markets. This paper will describe how SDL, the leading ITU standard language, can ideally extend the semantic coverage of the UML for modeling distributed and complex real-time systems, in particular to design the application architecture and give a dynamic semantics to the model. The constructs specialized for real-time software development are specified using the UML concepts. The resulting solution is illustrated by the presentation of the ObjectGEODE tool from CS VERILOG which offers a new and powerful combined UML/SDL approach while preserving current investment and culture. The presentation concludes on the works being led by the ITU and OMG.

Keywords: Real-Time (RT), SDL, UML, concurrent object, reuse, inheritance

Reminder on UML, Application Scope, Needs for an RT Profile

The UML [UML97] is an object-oriented modeling technique suited for the analysis and development of information systems for which the Class diagram is the core model. Additional diagrams, mainly Use cases, Sequence diagrams and State diagrams, help defining the dynamic behavior of the system and its objects, therefore making UML applicable to Real-Time (RT) development at a first sketch. But at a detailed level, UML has some weaknesses in this area:

- *Dynamic semantics:*

The resulting semantics of a complete model made of Class, Sequence, Collaboration, State and Activity diagrams is not precisely known: diagrams are partially redundant which could lead to introduce inconsistencies in some places.

You can also mix different interaction mechanisms within the same application, e.g. non-blocking asynchronous interaction, blocking synchronous interaction. What is the resulting global behavior?

In this paper we explain the dynamic semantics which is given to UML models to make them executable: how events are received then processed, how synchronous interactions run with asynchronous interactions, etc.

- *Action language:* There is not yet any standardized Action language associated to UML diagrams to formally describe transition actions or operation bodies. This is however required when a user wants to test the model before its implementation and also for automatic code generation. As a consequence we introduce the reader with a complete Action language dealing with object life cycle, communication, time and data computation.
- *Behavioral reuse:* A key requirement leading to select UML as the preferred design technique is the ability to support reuse and redefinition. However the resulting behavior of inheritance dependencies between hierarchical state diagrams is not precisely known in the UML specification. In addition, usual UML objects provide operations which can be inherited and redefined locally to match current application needs. In case of objects defined by a state machine, the redefinition of their behavior can also require to add new states and to change transition flows when for example new objectives are assigned to the objects. All the necessary construction rules added to UML for intensive behavioral reuse are specified in the paper.

We will use here the terms *active classes* for classes whose UML attribute `isActive` is equal to true, and *active objects* for instances of active classes. The engineering technique described here focuses on these elements.

The formal foundation for the UML profile¹ we present –informally– in this paper is based on the latest version of the SDL notation, a field-proven modeling language standardized by the ITU-T and widely used by telecom equipment manufacturers and telecom operators. This notation

¹ A UML profile corresponds to a specialization of the UML where semantic rules are added to the standard constructs in order to fulfil specific needs, for example, in our context, to design complex real-time systems.

provides software developers with all the OO concepts needed for real-time development: active classes, interfaces, packages, safe combination of asynchronous and synchronous interactions, formally defined inheritance between active classes, redefinition of transition. With these specialized constructs, software developers can so build adaptive reactive objects and generic architectural patterns, and reuse them by specializing their behavior. The resulting applications can be statically checked, can be tested and dynamically validated against formalized requirements, and can be converted to executable code which can run both on host and target platforms. The ObjectGEODE toolset gives a complete support to these needs.

Foundation of a UML Profile for RT Development

This section presents the foundation for the UML profile we propose for modeling complex RT applications, in order to give the appropriate power of expression and semantics to the standard UML in the areas of architecture, communication, executable behavior. This profile is based on the SDL notation which is precisely defined in [Z.100] and which is presented from a user point of view in [Ols94] and [Eli97]. The specialized constructs constituting this profile are presented according to their corresponding native UML constructs.

We do not detail here all the tagged values and constraints which are added to the stereotypes of the profile. This will require too much space. However a complete description of the semantics used as a basis here can be found in [Z.100].

The way these specialized constructs are used to design real-time software and what is their current representation are detailed in next section “RT Modeling by Example”.

Architecture and Communication

Active entity

An active entity is an active class. There are three kinds of active entity: *process type*, *block type* and *system type*—

- A process type is an active class which contains a state machine (detailed in sub-section “Behavior”).

ProcessType is a stereotype of Class (class defined in the UML metamodel) with constraint:



`isActive = true`. Its iconic representation is:

- A block type is an active class which is refined into block types or process types.

BlockType is a stereotype of Class with constraint: `isActive = true`. Its iconic



representation is:

- A system type is an active class corresponding to a whole application. A system type is refined into block types and cannot be included in any other active class.

SystemType is a stereotype of *SubSystem* (class defined in the UML metamodel) with constraint: `isInstantiable = true`. There is no iconic representation attached to this stereotype (*SystemType* is the top-level diagram).

Block types and system types can be used as generic design patterns: new block types and system types can be built from existing ones by inheritance. The inherited refinement can be modified: the active classes contained in the inherited active class can be modified and new active classes can be added. In addition, active entities can have contextual parameter (detailed below in item “Genericity”).

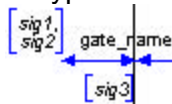
An active entity has interfaces through which it interacts with its environment by sending and receiving asynchronous or synchronous events. The environment cannot directly access to the classes contained in the container class, and the contained classes may interact with the environment only through the interfaces of the container class. Thus modularity is enforced making reuse safer and easier.

Gate

A gate is an interface on which are declared incoming and outgoing events. Events can be signals and remote procedure calls – detailed below. Gates are attached to active entities.

When an active class is inherited, all the gates of the superclass are gates of the subclass. These gates can be enriched by adding new events, new gates can also be added to the subclass.

Gate is a stereotype of *Interface* (class defined in the UML metamodel). Its iconic representation



is an arrow:

Signal

A signal is an asynchronous event. It contains the identity of the sender object (detailed in item “Active object”), and it may contains additional parameters.

Signal corresponds to class *Signal* already defined in the UML metamodel. Signals are described in notes, they have no specific iconic representation.

Procedure

A procedure is a subactivity of a state machine which is declared within a process type. Procedures can be called locally – the call is instantaneous –, or remotely – the call request is stored in the input queue of the receiver object and will be processed when all the events received before have been completely processed. Procedure calls are blocking actions: the caller is suspended until the procedure is completed and the return value (if any) is produced. When the procedure call is remote so-called RPC, this corresponds to a synchronous event. The RPC mechanism is useful to design client-server interactions (the server object provides a set of public procedures that are called by clients) and CORBA applications.

By default, procedures are hidden in the active classes and so cannot be called remotely (thus prohibiting non-visible object coupling).

If a procedure has states and inputs, then it shares the input queue of its container object.

A procedure can be inherited and redefined. Redefinition is a controlled mechanism: in the superclass, the procedure must be declared as *virtual* to allow its redefinition, and in the subclass, the redefined procedure must be declared as *redefined*. If a procedure is declared as *finalized*, it is no longer possible to redefine it.

Procedure is a stereotype of Submachine (class defined in the UML metamodel). Its iconic



Package

A package is a construct which contains declarations of types: active classes, signals, procedures and data types (detailed below in sub-section “Behavior”). When a package is used, all its declarations are accessible (selective importation is also possible).

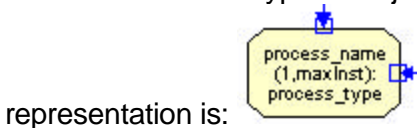
Package corresponds to class Package defined in the UML metamodel. There is no iconic representation defined so far in the profile; their use is declared by means of notes.

Active object

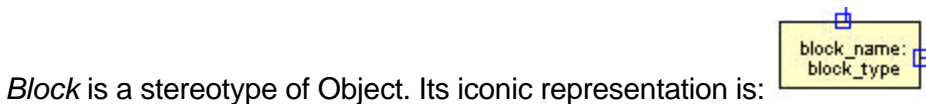
An active object is an instance of an active entity. There are three kinds of active objects: process, block and system—

- A process is an instance of process type. It can be created dynamically and be destroyed by suicide. It has a unique identifier. It runs the state machine defined in its related process type. It manages a single input queue storing chronologically the input events not yet processed.

Process is a stereotype of Object (class defined in the UML metamodel). Its iconic



- A block is an instance of block type. It is instantiated statically at the system start-up; a block type can be instantiated several times.



- A system is an instance of system type. It is instantiated statically once at the system start-up.

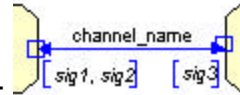
System is a stereotype of Instance (class defined in the UML metamodel) with the constraint that it is an instance of SubSystem. There is no iconic representation attached (System is the top-level diagram).

Channel

A channel is an instance of association between two gates of active objects. A channel conveys the signals and RPCs between the objects. Objects linked by a channel can be either two

objects of the same level, i.e. refining a same object, or the contained object and one of its container objects.

Channel is a stereotype of Instance with the constraint that it is an instance of Association (class



defined in the UML metamodel). Its iconic representation is:

Genericity

Active classes can have parameterized elements which must be defined at instantiation time, such classes correspond to templates. This mechanism also called context parameters here allows developers to build generic design patterns better reusable.

In the metamodel, an active class with context parameters corresponds to a template. Its iconic representation is the icon of its basic class (process type, block type or system type).

Behavior

State machine

A state machine is a set of states and transitions with a common input queue storing in the chronological order the incoming events, i.e. signals and RPCs.

StateMachine corresponds to class *StateMachine* defined in the UML metamodel, with additional semantics and constraints, mainly: one single input queue storing synchronous and asynchronous events, no concurrent states, no history. It has no iconic representation.

Transition

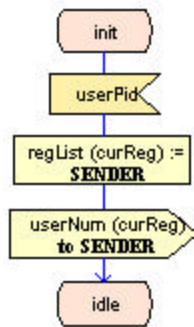
A transition has two parts: the guard and the body. The guard can disable the consumption of an input event. The body is a sequence of actions.

Compared to UML RT/ROOM [Sel94] [Sel98] or Shlaer-Mellor [Shl92], we propose a complete visual Action language to detail the content of transition actions, mainly: *signal output*, *timer set/reset*, *task* (contains expressions and access to class attributes and operations, see below item "Data types, variables and parameters"), *procedure call*, *process instance creation*, *decision*. The most used visual constructs are presented in next section "RT Modeling by Example".

Unexpected signals are removed from the input queue: when a signal is at the first rank in the input queue and there is no transition to consume it in the current state of the active object, then the signal is removed from the queue (*discarded*), and the next signal becomes ready to be processed.

Transitions can be inherited and redefined, similarly to procedures, using keywords *virtual*, *redefined* and *finalized*.

RTTransition is a stereotype of *Transition* (class defined in the UML metamodel) with additional semantics and constraints defined in the Action language sketched above. Its visual representation is made of a sequence of icons which represent the different actions of the transition body.



Example:

Data types, variables and parameters

Process and procedures can have local variables to store information. Signals, processes and procedures can also have parameters. Variables and parameters must be typed. A set of predefined types is defined for this purpose: *boolean*, *integer*, *real*, *charstring* (character string), etc.

Furthermore, in order to manipulate more complex data, type generators are provided to build arrays (*array*), lists (*string*) or structures (*struct*). New types so-called *Abstract Data Type (ADT)* correspond to classes that are not active: they contain attributes and operations but no state machine. Non-active classes can have parameterized elements, allowing the developers to build data templates.

Class operations can be called from transition actions. This is also a safe mechanism to integrate external code: external functions are encapsulated in ADT operations and can be so safely called from transitions.

ADT is a stereotype of Class with constraint: `isActive = false`.

Variable is a stereotype of Instance.

Parameter corresponds to class Parameter defined in the UML metamodel.

ADT, *Variable* and *Parameter* have a textual representation only, they are declared in notes.

Case Study

In order to show the use of the RT-specific profile, we have selected a realistic case study, obviously simplified for our purpose. It is based on an SDL tutorial developed at the 8th SDL Forum by Mr. Yair Lahav and Oystein Haugen [SDL97].

The studied problem takes its inspiration from a local exchange system. The characteristics of the system so-called *localExchange* are:

- It supports up to 9 local lines, numbered from 1 to 9.
- User events coming from the phone lines are: *offHook*, *onHook* and *num*.
- Events transmitted to the phone lines are: *dialTone*, *busyTone*, *shortBusyTone*, *ringTone*, *connectTone* and *msg*.
- After 30 seconds of *connectTone*, the communication stops.

- After 10 seconds of *busyTone*, *shortBusyTone* is transmitted.
- In basic configuration, digit 0 is not used.

The same system can also be upgraded to manage the CCBS phoning service, CCBS stands for *Compilation Call of Busy System*. In this enhanced configuration:

- When a 'caller' A dials a user B and the 'called' line B is busy, user A can dial 0 and then 8. The call will be registered, and user A will be re-connected when the line B is free again.
- When the local exchange system receives an 'hook on' signal from phone B, it rings phone A, and when A hooks off, it rings at B side.

Remarks: The same user can registered up to 5 calls. A user can remove all his registered calls by dialing 0 then 9. When user A hooks on before the B side hooks off, the registered call is removed from the queue.

RT Modeling by Example

Diagrams

Diagrams users made to design their application with this RT-specific UML profile are the followings:

- *Package diagrams*: Data types, events, active entities are declared. Data types and events are declared in notes using a formal textual syntax, active entities are defined using their iconic representations.
- *Collaboration (or Interconnection) diagrams*: compared to the standardized representation, RT Collaboration diagrams do not exhibit the event sequence numbering. This numbering is better captured and understandable in Sequence diagrams (also called MSC). These simplified Collaboration diagrams are so more legible and more general as they are no more dependent on one single event sequence.
- *Transition diagrams*: with the Procedure construct, they are conceptually equivalent to bubble-arrow State diagrams, but their representation puts the focus on transitions instead of states: transitions are graphically detailed by means of visual symbols.

We will see these different kinds of diagrams on the example described before. The solution used here has been presented by Mr. Yair Lahav² at the 8th SDL Forum [SDL97].

We start first with the basic configuration of Local Exchange system, without CCBS service. Then we will illustrate the powerful behavioral reuse capabilities by developing the CCBS service on top of the basic configuration.

Packages and Active Entities

Package diagram in fig. 1 shows textual declaration of types, signals (asynchronous events) and public procedures (synchronous events).

² Yair Lahav, ECI Telecom, Israel, yairl@ecitele.com

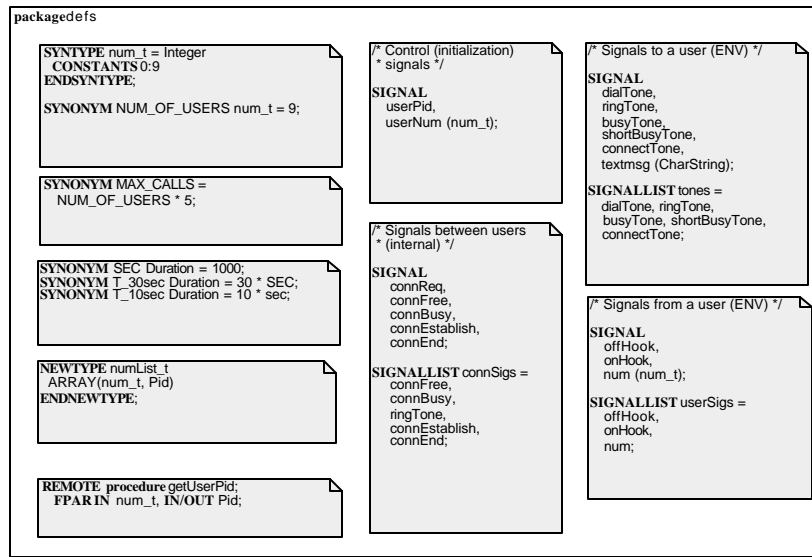


Fig. 1: Diagram of Package Defs

Package Defs contains the necessary declarations of constants –NUM_OF_USERS, MAX_CALLS...–, data types –num_t, numList_t–, signals and signallists –connReq, connFree, connSigs...– and public procedure getUserPid. These elements will be used mainly to formally defined the transition actions.

Active entities are declared in separate packages, see fig.2. Two active entities, userHandler_bt and switch_bt, are created to implement the system features.

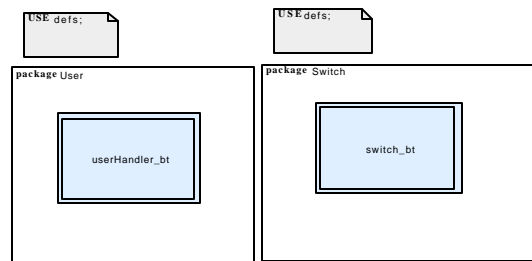


Fig. 2: Diagrams of Packages User and Switch

Fig. 3 shows the content of active entity switch_bt. This entity contains two other active entities, switchCtrl_pt –to monitor incoming communication requests– and connHandler_pt –to monitor communications between two users. These entities are no more refined (use of ProcessType icon). Entity switch_bt also contains 2 interfaces CG and CiG, and several instances: switchCtrl is an instance of switchCtrl_pt and there is a set of instances connHandler(0,MAX_CALLS) of type connHandler_pt, each connHandler object manages an ongoing phone communication, connHandler objects are dynamically created from switchCtrl.

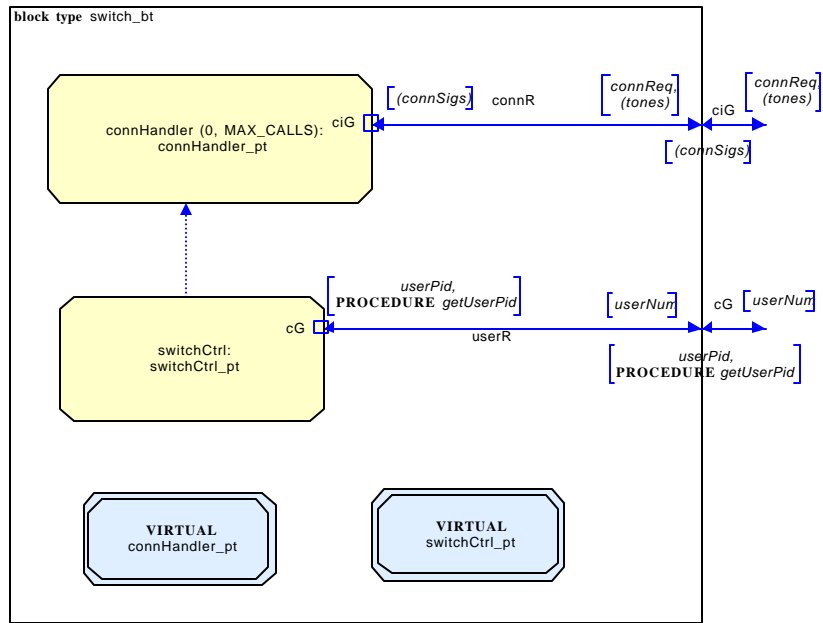


Fig. 3: Collaboration diagram of Active entity switch_bt

Active entity userHandler_bt is not detailed in this paper.

Architecture of the Application

The top-level Collaboration diagram of the Local Exchange system is shown in fig. 4.

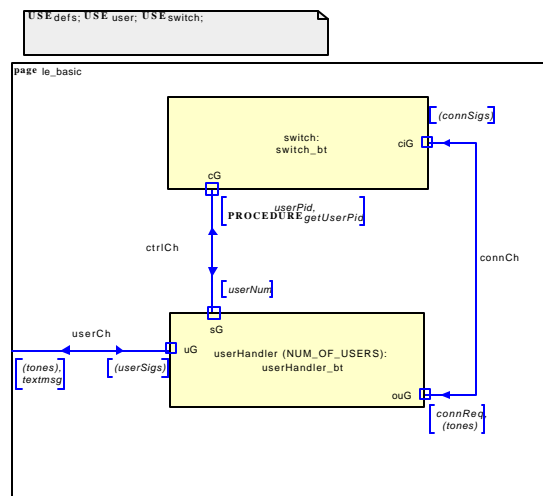


Fig. 4: Top-level Collaboration diagram of Local Exchange

The Local Exchange system is made of one instance of active entity switch_bt –the object switch monitors all the ongoing user communications– and a set of instances of active entity userHandler_bt –each object userHandler monitors one user phone line. The communication

links are precisely described: no other interactions can occur that do not conform to these links. Gates of the reused active entities are connected through channels, either to the environment, or to other internal objects; channels and gates must be consistent.

Transition Diagrams

Let's have a look at active entity connHandler_pt whose instances manage established communications. Its Transition diagram is given in fig. 5.

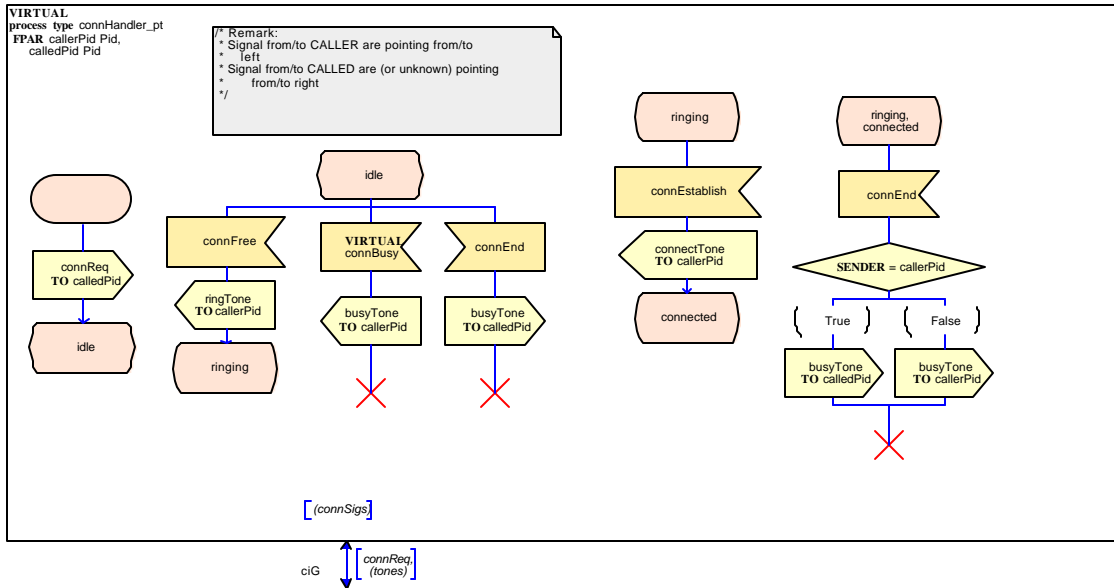


Fig. 5: Transition diagram of connHandler

This active entity has one interface `ciG`. Its state machine contains three states, `idle` –start of the call–, `ringing` –phone is ringing– and `connected` –users are in dialog. From `idle`, `connHandler` processes the three incoming signals: `connFree` –the called line is free–, `connBusy` –the called line is already connected–, `connEnd` –communication must stop. In state `ringing`, if the call request is accepted –`connEstablished`–, the state machine switches to state `connected`. The communication stops on reception of `connEnd`, and the current instance commits suicide.

In this diagram, transition actions are formally defined based on data types, object identifiers, signals, class operations, parameters, etc.

Development of the CCBS Service by Inheritance and Redefinition

The development of the CCBS service in the Local exchange system is made by redefining the active entities forming the initial version of the design model.

First, a new package is created to declare all the events and types related to the CCBS service; examples of new CCBS events are `pendRingTone`, `pendReq`, `pendAccept`.

Then, new classes `switch_CCBS_bt` and `userHandler_CCBS_bt` are created based on the existing classes `switch_bt` and `userHandler_bt`: the new ones inherit from the second ones their interfaces and contained classes. As an example, fig. 6 shows the definition of `switch_CCBS_bt`:

- A new interface vG is added to the two existing interfaces cG and ciG, in order to convey the CCBS events.
- The two active classes switchCtrl_pt and connHandler_pt are redefined to implement the processing linked to the CCBS events, and their instances are connected to the interface by means of new links.

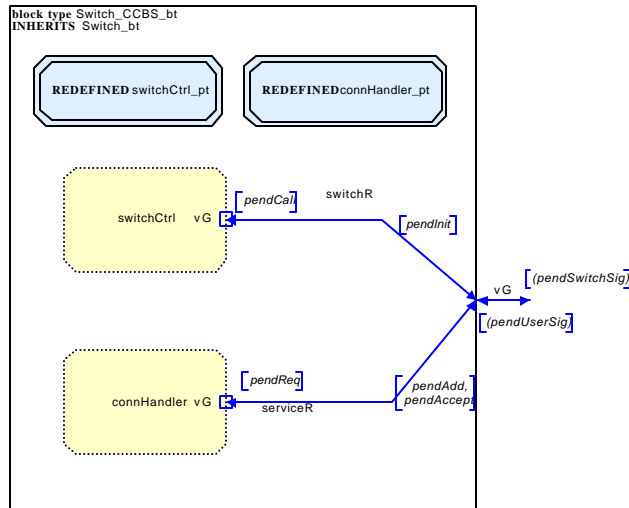


Figure 6: Definition of `switch_CCBS_bt` based on `switch_bt`

Fig. 7 shows the redefinition of `connHandler_pt`. All the existing interfaces and transitions are inherited in the new subclass. There is one transition redefined: reception of `connBusy` in state `idle` (keyword `REDEFINED` in the Input symbol). The new elements are: an interface is added to convey the CCBS events (seen in fig. 6); a new variable `plnfo` is added to store the pending calls; a new state is created `CCBS_or_onHook` to process the CCBS event `pendReq` (request for pending call). This state and its transitions complement the initial set of states and transitions.

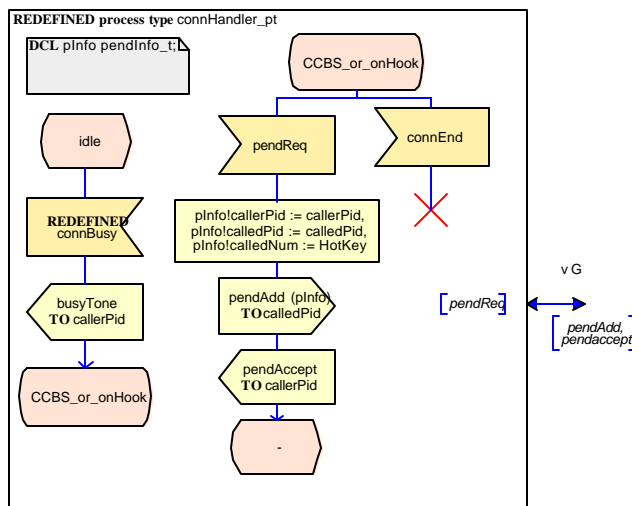


Figure 7: Redefinition of `connHandler_pt`

The new Local exchange software is now obtained by instantiating the `switch_CCBS_bt` and `userHandler_CCBS_bt` classes, this constitutes the top-level design diagram (similarly to fig. 4).

ObjectGEODE Support

The RT-specific UML profile described here is already fully supported by the ObjectGEODE toolset developed by CS VERILOG. The toolset includes, see fig. 8: graphical editors for Class diagram, State charts and specialized versions of Package diagram, Collaboration diagram, Transition diagram and Sequence diagram (MSC); a tool to test and validate RT design models; a tool to generate test cases from RT design models; two code generators for the production of C++ skeleton code from UML Class diagram and 100% executable C source code from RT design models that can be deployed on real-time target platforms; a tracing tool to help designers debug RT models on target platforms.

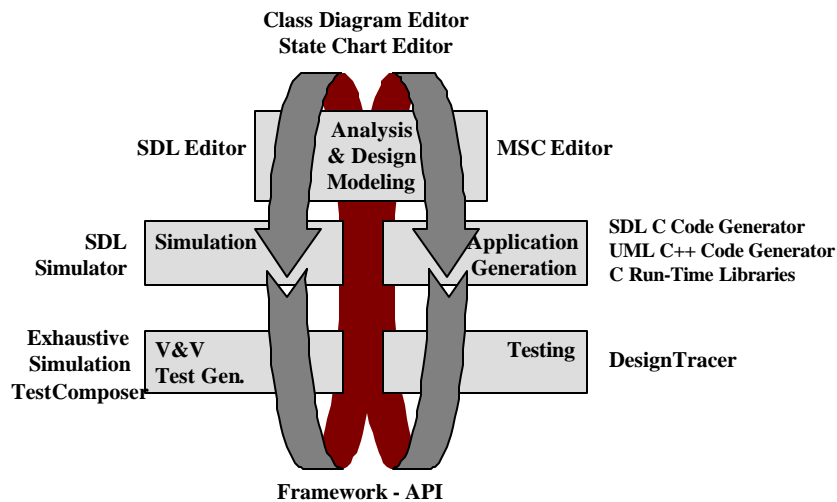


Fig. 8: ObjectGEODE Toolset Composition

The first benefit of the RT-specific UML profile used jointly with the ObjectGEODE tools is that designers build models that can be executed according to the dynamic semantics defined in the profile. Designers can validate their logical behavior early in the life cycle. Executable code can be generated reducing implementation costs.

The visual Action language is totally supported throughout the toolset at the modeling (Transition diagram editor), testing and autocoding levels. This greatly increases the easiness-of-use of formal design techniques as designers have not to be experts in cumbersome textual syntaxes.

The third major benefit results from the reuse capabilities which are extended compared to the standard UML (and the way it is usually supported by commercial tools). In accordance with the RT-specific profile, the ObjectGEODE toolset checks that active entities are correctly reused and redefined. As a consequence misuse is detected both from a static point of view by type checking (data and structural entities), and from a dynamic point of view, by model simulation and validation.

ObjectGEODE is the ideal toolset for an iterative development process. At each iteration, a part – *increment*– of the application is designed, this increment can then be tested and verified against requirements both at design level with the simulation tool, as well as at code level with the autocoding tool (on host or target platforms). Once the increment is completed, we start a new iteration, the new increment will be added to the existing ones.

Ongoing Work on UML and SDL

There is now a general consensus in the OMG to say that UML (1.3) does not fully match the constraints regarding real-time software engineering. Several RFPs are currently opened to elaborate adequate extensions, and some UML profiles are being developed (one profile being *UML RT* which consists in merging UML 1.3 and ROOM [Sel94], its foundation is presented in [Sel98]).

In parallel, the ITU-T standardization body is working on a complete and formal definition of an SDL-based UML profile. This work is carried out by the SDL working group *Study Group 10 – Question 6* chaired by Mr. Rick Reed (TSE Ltd, UK), and under the coordination of Mr. Birger Møller-Pedersen (Ericsson, Norway). This will result in a version of the SDL Recommendation fully compatible with UML for 2000. They are operating in 3 steps:

- First, SDL is being enhanced in order to make its correspondence with UML easier. For example, concepts of SDL active class, interface, state and associations are extended.
- Then, a formal mapping is being given between the SDL constructs and the native UML constructs. The complete definition of the SDL-based UML profile will be provided in a separate ITU-T document [Z.109]. The approved version of this document will be issued by the ITU-T next year.
- Lastly, the SDL graphical representation is being enriched with the UML graphical representation each time UML provides the corresponding graphical construct. This will be part of the next approved version of the Z.100 ITU-T Recommendation to be published next year. Therefore, UML designers will be familiar with SDL-2000 diagrams and traditional SDL designers will be still allowed to use the traditional SDL representation.

This ITU-T definition work is conducted in coherence with the OMG work on the UML evolution through active participation of Q6/10 experts.

Acknowledgements

We would like to thank the ITU-T group Q6/10 for his tremendous work which constitutes the foundation of the approach presented here, in particular: Rodolphe Arthaud, Ileana Ober, Rick Reed, Birger Møller-Pedersen, Thomas Weigert, Eckhardt Holz, Andreas Prinz, Ralf Shröder, Anders Ek and Anders Olsen.

References

- [Eli97] Ellsberger J., Hogrefe D., Sarma A., *SDL, Formal Object-oriented Language for Communicating Systems*, Prentice Hall Europe, 1997.

- [Ols94] Olsen A., Faergemand O., Moller-Pedersen B., Reed R., Smith J.R.W., *Systems Engineering Using SDL-92*, North Holland, 1994.
- [SDL97] SDL'97 (8th SDL Forum), *Tutorials*, Ana Cavalli - Daniel Vincent editors, INT, 1997.
- [Sel94] Selic B., Gullekson G., Ward P., *Real-time Object-Oriented Modeling*, John Wiley & Sons, Inc., 1994.
- [Sel98] Selic B., Rumbaugh J., *Using UML for Modeling Complex Real-time Systems*, Rational white paper, www.rational.com, 1998.
- [Shl92] Shlaer S., Mellor S., *Object-Oriented Systems Analysis, Modeling the World in States*, Yourdon Press, Prentice Hall, 1992.
- [UML97] Object Management Group (OMG), *UML 1.1 Documentation Set*, ftp.omg.org/pub/docs/ad, 1997.
- [Z.100] ITU-T, Recommendation Z.100, Specification and Description Language (SDL), <http://www.itu.ch> - Electronic Bookshop, Geneva, 1996.
- [Z.105] ITU-T, Recommendation Z.105, SDL Combined with ASN.1 (SDL/ASN.1), <http://www.itu.ch> - Electronic Bookshop, Geneva, 1995.
- [Z.109] ITU-T, Draft document, SDL-UML, SG.10 Q.6, Editor Birger Moller-Pedersen, 1999.
- [Z.120] ITU-T, Recommendation Z.120, Message Sequence Chart (MSC), <http://www.itu.ch> - Electronic Bookshop, Geneva, 1996.