UML and RT-LOTOS
An Integration for Real-Time System Validation

P. de Saqui-Sannes*,** — L. Apvrille***,**** — C. Lohr** — P. Sénac*** — J.-P. Courtiat

*ENSICA, 1 place Emile Blouin, 31056 Toulouse Cedex 05, France
{desaqui, apvrille, senac}@ensica.fr
**LAAS-CNRS, 7 avenue du Colonel Roche, 31077 Toulouse Cedex 04, France
{lohr, courtiat}@laas.fr
***Alcatel Space Industries, 26 avenue J.-F. Champollion, B.P. 1187, 31037 Toulouse Cedex 01, France

ABSTRACT. The paper presents a UML profile that overcomes the limitations of real-time solutions currently available on the market. Associations between classes are given a formal semantics. New temporal operators are introduced; they include a non deterministic delay and a time-limited offering. UML models can be validated against logical and timing constraints. The profile’s semantics is given through a translation into the formal language RT-LOTOS. The latter is supported by a validation tool which generates reachability graphs from extended UML models. A coffee machine serves as example in the paper. The profile is under evaluation on a satellite-based software reconfiguration system.


RÉSUMÉ. Face aux limitations des solutions UML temps réel actuellement sur le marché, l’article présente un profil UML qui donne une sémantique formelle aux associations entre classes, définit des opérateurs temporels de type délai non déterministe et d’offre limitée dans le temps et ajoute des facilités de validation de contraintes logiques et temporelles. La sémantique formelle de ce profil est donnée par la traduction dans le langage formel RT-LOTOS dont l’outil de validation permet de construire des graphes d’accessibilité à partir de diagrammes UML étendus. Outre l’exemple de la machine à café traité dans l’article, le profil proposé est en cours d’évaluation sur un système de reconfiguration dynamique de logiciel embarqué à bord de satellite.

MOTS-CLÉS : Systèmes temps réel, Méthodes formelles, UML, RT-LOTOS, Validation.
1. Introduction

With the notion of profile, the OMG-based Unified Modeling Language [OMG 01] has been defined as a general purpose modeling language that can be specialized for specific domains. Before a real-time profile specification was released at OMG [OMG 02], several companies have competed to propose proprietary “Real-time UML” solutions [SEL 98, ART 99, DOU 99, EST 02]. Meanwhile, the need for an enhanced UML with real-time features has stimulated research work on integrating UML and Formal Description Techniques that had already been applied to time-critical systems [DEL 98, CLA 00, AND 01, DUP 01].

The TURTLE\(^2\) profile [SAQ 01] presented in the paper extends UML with concepts borrowed from the Formal Description Technique RT-LOTOS\(^3\) [COU 00]. Class diagrams are modified so that parallelism and synchronization between classes can be expressed explicitly. Extended activity diagrams with a non deterministic delay and a time-limited offering are used instead of Statecharts to describe classes’ internal behaviours. Class and activity diagrams are translated into RT-LOTOS, and the resulting specification is provided as input to the RTL\(^4\) tool. This makes it possible to perform a priori validation on TURTLE diagrams by checking models against logical and timing errors.

The paper is organized as follows. Section 2 surveys related work. Section 3 introduces RT-LOTOS. Section 4 defines the TURTLE profile. Section 5 discusses its application to a coffee machine. Section 6 concludes the paper.

2. Related work

Several tool manufacturers have competed to offer real-time UML solutions with an enhanced notation and a methodology:
- Rose RT implements UML-RT, an enhanced UML with concepts from the ROOM language [SEL 98];
- Rhapsody by I-Logix uses as much as possible native UML 1.4 constructs [DOU 99];
- TAU by Telelogic uses UML as a front-end for SDL [BJO 00];
- Real-Time Studio [ART 99] by Artisan Software has its own temporal operator;
- Esterel Studio [EST 02] by Esterel-technologies combines UML and synchronous language Esterel.

---

1 A UML profile specializes the UML meta-model into a specific meta-model dedicated to a given application domain [TER 00]. A profile may contain selected elements of the reference meta-model, extension mechanisms, a description of the profile semantics, additional notations, and rules for model translation, validation and presentation.
2 Timed UML and RT-LOTOS Environment.
3 Real-Time LOTOS (Language Of Temporal Ordering of events).
4 RT-LOTOS Laboratory.
The first four tools in the list above implement an asynchronous paradigm. They also share in common temporal operators limited to timers with a fixed duration. They miss native operators to express time interval and time-limited actions within behavioural diagrams. When solutions nevertheless exist, they remain oriented towards code generation for a specific target and operating system. A priori and implementation-independent validation of UML models cannot be carried out.

On the academic side, a lot of work has been done on providing UML with a precise semantics [BRU 98, BRU 99, EVA 99] and connecting UML with a Formal Description Technique, such as Labeled Transition Systems [JAR 98, GUE 00], Petri Nets [DEL 98], Z [DUP 01], synchronous languages [AND 01], PVS [TRA 00] and ELOTOS [CLA 00]. Unlike [DEL 98], the profile in Section 4 remains UML 1.4 compliant in the way it integrates concepts borrowed from the RT-LOTOS FDT. The latter is an asynchronous language, which differs from [AND 01]. Like [DUP 01], [TRA 00] and [CLA 00], the translation procedure from extended UML to RT-LOTOS gives a formal semantics to the profile. Major differences between RT-LOTOS and ELOTOS include a non deterministic delay operator (see the latency operator in Section 3) and validation techniques implemented by a tool.

3. RT-LOTOS

LOTOS [BOL 87] is an ISO-based Formal Description Technique for distributed processing system specification and design. A LOTOS specification, itself a process, is structured into processes. A LOTOS process is a black box which communicates with its environment through gates using multiple rendezvous. Values can be exchanged at synchronization time. Exchanges can be mono- or bi-directional.

Parallelism and synchronization between processes are expressed by composition operators. The latter include process sequencing, synchronization on all communication gates and synchronization on some gates, a non deterministic choice and interleaving (parallel composition with no synchronization). Composition operators are identified by their symbols (Table 1).

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ ]</td>
<td>Choice.</td>
<td>P[a,b,c,d] = P1[a,b] [ ] P2[c,d]</td>
</tr>
<tr>
<td>[ ]</td>
<td>Parallel composition with no synchronization.</td>
<td>P[a,b,c,d] = P1[a,b] [ ] P2[c,d]</td>
</tr>
<tr>
<td>[b,c,d]</td>
<td>Parallel composition with synchronization on several gates (b,c,d).</td>
<td>P[a,b,c,d,e] = P1[a,b,c,d] [b,c,d] P2[b,c,d,e]</td>
</tr>
<tr>
<td>hide b in [b]</td>
<td>Parallel composition with synchronization on gate b, which is hidden.</td>
<td>P[a,c] = hide b in P1[a,b] [b] P2[b,c]</td>
</tr>
<tr>
<td>&gt;&gt;</td>
<td>Sequential composition.</td>
<td>P[a,b,c,d] = P1[a,b] &gt;&gt; P2[c,d]</td>
</tr>
<tr>
<td>[&gt;]</td>
<td>Disrupt (P2 preempts P1).</td>
<td>P[a,b,c,d] = P1[a,b] [&gt;] P2[c,d]</td>
</tr>
</tbody>
</table>
RT-LOTOS extends LOTOS with three temporal operators (Table 2). The combination of a deterministic and a non deterministic delay makes it possible to handle time intervals. RT-LOTOS reuses and extends the control part of LOTOS, but replaces algebraic data types by implementations in C++ or Java [COU 00].

<table>
<thead>
<tr>
<th>Temporal operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>a{t}</td>
<td>Time limited offering.</td>
</tr>
<tr>
<td>delay(t)</td>
<td>Deterministic delay.</td>
</tr>
<tr>
<td>latency(t)</td>
<td>Non deterministic delay.</td>
</tr>
</tbody>
</table>

Table 2. RT-LOTOS temporal operators

4. TURTLE profile

The TURTLE profile enhances class and activity diagrams using the extension mechanisms allowed by UML 1.4, in particular stereotypes\(^5\). Thus, a TURTLE class diagram contains “normal” classes and classes stereotyped as Tclass. Two classes can be linked by one of the four following relationships: use, aggregation, composition, and generalization.

TURTLE class diagrams introduce two important features: first, two Tclasses can synchronize on so-called Gates; second, associations between two Tclasses can be attributed by a composition operator. TURTLE activity diagrams offer new symbols, in particular temporal operators inherited from RT-LOTOS ones (Table 2).

4.1. Gate Abstract Type

Two Tclasses can communicate using input and output Gates. A Gate abstract type (Fig. 1a) serves as super-type for InGate and OutGate, respectively (Fig. 1b).

\(^5\) A stereotype is an indirect addition to the meta-model. The TURTLE stereotype and abstract types are graphically identified by a “turtle symbol” in the upper right corner of the class.
In the paper, we say that “a Tclass performs an action on Gate g” to express that the Tclass wants to communicate on Gate g.

4.2. Tclass Stereotype

A Tclass stereotype is a UML class with two basic constraints: Gates are separated from other attributes, and the behaviour description must be an activity diagram (Fig. 2). Other properties to be satisfied by a Tclass are listed in [SAQ 01].

![Table](Tclass components)

**Figure 2. Tclass components**

4.3. Composer Abstract Type

A UML class diagram graphically defines a set of classes interconnected by relationships, in particular associations. TURTLE further makes it possible to give an association a precise semantics. The Composer abstract type is introduced to support that idea. Note that Composer is not used directly; associations are attributed with so-called “associative” classes (Parallel, Synchro, Invocation, Sequence, Preemption) that inherit from Composer. Two inherited classes of Composer are presented in Fig. 3.

![Diagram](Use of two inherited classes of the Composer abstract type)

**Figure 3. Use of two inherited classes of the Composer abstract type**
For each association between two Tclasses, there exists one and only one meaning, and therefore one Composer. Let us now review the classes which inherit from Composer.

**Parallel**

Two Tclasses related by an association assigned by the Parallel operator are executed in parallel, and without any synchronization. The two Tclasses should be active\(^6\) classes.

**Synchro**

Two Tclasses related by an association attributed by the Synchro operator can synchronize with each other. This synchronization is executed by the two Tclasses in two separate execution threads. A synchronization possibly includes a data exchange; inputs and outputs are detailed in the respective behaviour descriptions of the two classes involved in the synchronization. If the association between the two Tclasses includes a navigation indication, the data exchange can only take place in the direction indicated by the navigation. Two Tclasses may synchronize on different Gates that must be listed in an OCL (Object Constraint Language) formula. For example, suppose that Gates g1 and g2 of Tclass T1 synchronize respectively with Gates g3 and g4 of Tclass T2; in that case, the OCL formula associated with the association should be \{T1.g1 = T2.g3 and T1.g2 = T2.g4\}. Each time T1 performs an action on g1, it must wait for T2 to perform an action on g3, and *vice versa*.

**Invocation**

Whereas the Synchro operator denotes a synchronization between two separate execution flows, Invocation denotes a synchronization which, like an operation call in the object paradigm, is performed within the caller’s execution flow.

Let us consider two Tclasses T1 and T2 linked by an association directed from T1 to T2 and attributed by the Invocation associative class. T2 can be activated by T1. Both T1 and T2 must have a Gate involved in the invocation. For example, let us consider that g1 (resp. g2) is a T1 (resp. T2) Gate and that the \{T1.g1 = T2.g2\} OCL formula is added to the association. Then, when T1 performs an action on g1, it must wait for T2 to perform an action on g2. When T2 performs an action on g2, data can only be exchanged as indicated by the navigation. T1 is then blocked on g1 until T2 performs again an action on g2. The second data exchange can only be performed from the callee to the caller.

**Sequence**

Two Tclasses related by an association to which this operator is associated are triggered one after the other in the association’s navigation direction. Note that in (T1 Sequence T2), T1 must terminate\(^7\) before T2 starts. The two Tclasses should be active classes.

---

\(^6\) A class is active if it represents an execution flow of the system [DOU 99].

\(^7\) A Tclass terminates when all its activities have reached their termination points.
**Preemption** A `Tclass` pointed by the navigation of an association attributed by the `Preemption` operator may interrupt the other `Tclass` at any time. In practice, "T2 Preemption T1" means that T2 may preempt T1, *i.e.* it kills T1 and activates T2.

### 4.4. `Tclass` Behavior Description

UML activity diagrams symbols are supported, but operation calls are not translated to RT-LOTOS, assuming that two `Tclasses` use gate synchronization to communicate. Table 3 lists all the symbols, and associates the relevant translations in RT-LOTOS; \( \tau(AD) \) denotes the translation process for the sub-diagram AD which follows the symbol.

<table>
<thead>
<tr>
<th><strong>TURTLE activity diagram</strong></th>
<th><strong>Description</strong></th>
<th><strong>LOTOS translation</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>![AD](g !x ?y)</td>
<td>Synchronization on Gate g, possibly with emission of value and/or reception. AD is subsequently interpreted.</td>
<td>( g \ x \ y ) ( \text{nat} ) ( \tau(AD) )</td>
</tr>
<tr>
<td><img src="AD" alt="y := x^2" /></td>
<td>Value assignment of an attribute. AD is subsequently interpreted.</td>
<td>let ( y : \text{YType} = x^2 ) in ( \tau(AD) )</td>
</tr>
<tr>
<td>![AD](or AD)</td>
<td>Loop structure. AD is interpreted each time the loop is entered.</td>
<td>process ( \text{LabelX}[g_1, ..., g_n] : \text{noexit} := \tau(AD) ) ( \text{&gt;&gt;LabelX}[g_1, ..., g_n] ) end proc</td>
</tr>
<tr>
<td>![AD1 AD2 ... ADn](g1, ..., gm)</td>
<td>Synchronization on Gates g1, ..., gm between n sub-activities described by AD1, AD2, ..., ADn. The gate list is possibly empty.</td>
<td>( \tau(AD_1) |M{[g_1, \ldots, g_m]}</td>
</tr>
<tr>
<td>![AD1 AD2 ... ADn](c1, c2, ..., cn)</td>
<td>Conditions are optional AD1, AD2, ..., ADn sub-activities for which conditions are true can be selected. One ready-to-execute activity whose condition is true is executed.</td>
<td>( [\text{c}_1] \rightarrow \tau(AD_1) ) ( [\text{c}_2] \rightarrow \tau(AD_2) ) ( \cdots ) ( [\text{c}_n] \rightarrow \tau(AD_n) )</td>
</tr>
</tbody>
</table>
The n sub-activities described by AD1, AD2, ..., ADn are followed by the execution of the m sub-activities described by AD’1, AD’2, ..., AD’m. The AD’i are executed with synchronization on k Gates g1, g2, ..., gk.

(τ(AD1) ||| τ(AD2) ||| ... τ(ADn) ) >>
(τ(AD’1) || [g1, ...gk] || τ(AD’2)) || [g1, ...gk] || ...
(τ(AD’m))

Termination of an activity.

Table 3. Non temporal TURTLE operators

Table 4 lists pictograms associated with the temporal operators which extend UML activity diagrams. The third operator applies to a time interval. It is equivalent to two operators put in sequence: first, a fixed duration delay equal to the interval’s lower bound, and second, a latency equal to the difference between the interval’s upper and lower bounds.

<table>
<thead>
<tr>
<th>TURTLE operator</th>
<th>Description</th>
<th>RT-LOTOS translation</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="d" alt="d" /> AD</td>
<td>Deterministic delay. AD is interpreted after d time units.</td>
<td>delay(d) τ(AD)</td>
</tr>
<tr>
<td><img src="t" alt="t" /> AD</td>
<td>Non deterministic delay. AD is interpreted at most after t time units.</td>
<td>latency(t) τ(AD)</td>
</tr>
<tr>
<td><img src="dmin" alt="dmin" /> dmax - dmin</td>
<td>Non deterministic delay between dmin and dmax. AD is interpreted at least after dmin and at most after dmax time units.</td>
<td>delay(dmin,dmax) τ(AD)</td>
</tr>
<tr>
<td><img src="a" alt="a" /> AD1 AD2</td>
<td>Time-limited offering. Action a is offered during a period which is less or equal to t. Note that latency and time-limited offering start at the same time. If the offer happens, AD1 is interpreted. Otherwise, AD2 is interpreted.</td>
<td>latency(t) a{t, τ(AD2)}; τ(AD1)</td>
</tr>
</tbody>
</table>

Table 4. TURTLE temporal operators
4.5. Validation Process

The TURTLE profile has been developed to validate real-time system models against design errors, and timing inconsistencies in particular. Figure 4 depicts the validation process. TURTLE classes and their relationships are extracted from the class diagram, saved under an XMI file, and converted into RT-LOTOS code which is validated using the RTL tool. Systems of reasonable size can be checked using reachability analysis techniques [COU 00]. Otherwise, simulation is limited to a partial exploration of the system’s behaviour.

Figure 4. From a TURTLE model to validation

5. Application: a coffee machine

The purpose of this section is to illustrate the TURTLE syntax, and to demonstrate the interest of using the RTL tool to discover logical errors and time constraints violations.

The TURTLE diagram in Fig. 5 models a coffee machine which distributes tea or coffee after two coins have been inserted by a user. The user has a wallet, not described for space reasons. One can notice the inheritance relation between CoffeeMachine and CoinBox as well as the synchronizations between CoffeeMachine and Wallet or Button, respectively.

Let us now comment on temporal operators used in Fig. 5. The role of time limited offering coinDelay in CoffeeMachine is to guarantee that a user who waits too much before inserting a second coin will get the first one back. Similarly, buttonDelay manages the situation where a user waits too much before selecting a drink. The deterministic delay delay in Button represents a button’s response time. Joint use of deterministic and non deterministic delays makes it possible to represent coffee and tea preparation times as temporal intervals: [100, 100+75] and [120, 120+80], respectively.

Let us now pay attention to synchronizations active1 and active2 in CoffeeMachine and time limited offering in Button. Both contribute to solve a problem identified in a simpler model [SAQ 01] of the coffee machine. Let us assume the user inserts two coins and waits too long. The synchronization offer on tea or coffee expires, which means that both coins are ejected. The user pushes the button tea (Button class, push Gate) just afterwards. The synchronization offer can no longer take place. The user takes his coins back, thinking the machine is out of
order. A user wishing to have a coffee arrives and inserts two coins. As the synchronization offer on \textit{tea} has not expired (unlimited offer), he or she is instantly served a tea. The problem is solved as follows: synchronization on \textit{active1} and \textit{active2} enables the machine to activate the two buttons for a limited period of time (\textit{push} offer limited to 40). Once the two buttons are activated, it still takes 50 ms (delay = 50) before the machine can synchronize on \textit{coffee} or \textit{tea}.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{TURTLE_class_diagram_for_a_coffee_machine.png}
\caption{TURTLE class diagram for a coffee machine}
\end{figure}

Logical and timing errors have been found using the RTL tool. For space reasons, Fig.6 depicts the reachability graph obtained for a machine limited to distributing tea. For each logical state (rectangle), several classes of temporal states (circles) may coexist. Conditions for leaving a state are as follows: either time has elapsed (transition $t$) or a synchronization has occurred. Let us take examples from the reachability graph in Fig.6a. Moving from the initial state (state 0) demands synchronization on \textit{Gate putCoin}. In state 21, no synchronization can occur in the first two sub-states; a state change corresponds to a time progression exclusively (transition $t$). When the offer on \textit{Gate tea} expires (delay \textit{buttonDelay}), then, a synchronization on \textit{coinBack} makes it possible to move from State 21 to State 7; a value equal to two is exchanged at that occasion.

The graph in Fig. 6a highlights that it is impossible for a user to get either tea or coffee. In fact, the button activation delay (\textit{push}) expires before the machine is ready to deliver coffee or tea. If this delay is increased from 40 to 60 (Fig. 6b), it becomes possible to get tea: the transition from state 21 is now \textit{tea}. 


6. Conclusions and Future Work

The paper defines TURTLE, a UML profile for real-time system design and validation. Class diagrams are extended with a stereotype ($Tclass$) and two abstract types ($Gate$ and $Composer$). A precise semantics is given to associations between classes (see the $Parallel$, $Synchro$, $Invocation$, $Sequence$ and $Preemption$ classes). The behaviour of a $Tclass$ is described by an enhanced activity diagram with three temporal operators: a deterministic delay, a non deterministic delay and a time-limited offering. Last but not least, TURTLE models can be translated into RT-LOTOS, a formal description technique supported by a validation tool. RT-LOTOS specifications derived from TURTLE diagrams can be validated using reachability analysis techniques. The objective is to keep RT-LOTOS hidden to the system designer.
The TURTLE profile is under evaluation on real-time embedded software. In particular, it is used for the formal validation of dynamic reconfiguration of embedded software [APV 01].

The TURTLE profile will be extended in the near future. State machines will be used in lieu of activity diagrams. New associative classes will be introduced to extend association semantics (resume/suspend, interrupt) [HER 98]. Our intent is to perform schedulability analysis on TURTLE models [AND 97]. Finally, relationships between the TURTLE profile and the OMG one [OMG 02] are under study.

7. References


