CHECKING BEHAVIOURAL CONSISTENCY OF UML-RT MODELS THROUGH TRACE-BASED SEMANTICS

Luis E. Mendoza Morales
Departamento de Procesos y Sistemas, Edificio de Matemáticas y Sistemas, Universidad Simón Bolívar
Apartado 89000, Baruta, Caracas, 1080-A, Venezuela

Manuel I. Capel Tuñón, Kawtar Benghazi Akhlaki
Departamento de Lenguajes y Sistemas Informáticos, ETSI Informática
Campus Aynadamar, Universidad de Granada, 18071 Granada, Spain

Keywords: Real-time software systems, UML-RT, Formal semantic, Formal specification, CSP+T.

Abstract: Starting from a methodological approach intended to obtain a correct system specification in CSP+T from a UML–RT model of an RTS, we develop now a systematic procedure to check whether the obtained design is consistent with other views of the same system, such as the ones given by class, composite structure and state machines diagrams. To achieve this objective, a formal semantics of the notational elements of UML–RT according to CSP+T process terms is presented, which guarantees that system requirements are preserved from their initial UML–RT modelling to the final system implementation. As a consequence, the formal support given by the compositional refinement of CSP+T process terms will allow performing the system's software compositional verification. In addition, the derived formal semantic definitions are applied to the Production Cell case study.

1 INTRODUCTION

The increase of the technological level of software development capabilities and the presence of computer systems everywhere has encouraged the deployment of real–time systems (RTS) to monitor and control many different kind of applications, which range from household appliances to control of chemical plants, nuclear power stations, command and control systems or the control of laboratory experiments. The complexity of modern real–time embedded control systems (RTECS) together with the absence of appropriate software tools is one of the reasons for the large number of errors in the design and implementation of these systems. Moreover, exhaustive testing is impossible, because of the combinatorial explosion of different runs that an RTECS can undergo when it executes.

Formal methods are the conceptual tools that allow the construction of reliable RTECS despite of their complexity. It has been pointed out by several authors (Evans et al., 1998; Kim and Carrington, 1999) that advancing towards the combination of semi–formal modelling languages, such as UML, thereby contributing to a more widespread use of rigorous Software Engineering methods in the development of these systems.

This article describes how to give a timed semantics to the UML–RT notational elements in terms of the CSP+T formal specification language (Žic, 1994), and establishes the consistency conditions of the models obtained by applying a transformational methodology (Benghazi et al., 2007).

Other contributions based on the formalization of the dynamic model describing the behaviour of different types of systems have been carried out in the OMT's dynamic model (Cheng and Wang, 2002), UML's state machine (SM) (Engels et al., 2001a) and sequence diagrams (Haugen et al., 2005). The first one formalizes (SMDs) by giving them a structured semantics based on LOTOS (Bolognesi and Brinksma, 1987). In the second one a formal trace semantics is given to SMs in the context of composite structure diagrams of UML 2.0 (OMG, 2004); thus, a semantic interpretation as CSP process terms is given to capsule–SMs, connectors and ports (Engels et al., 2001b). The third one describes STAIRS, an approach to the compositional development of UML's sequence diagrams resorting to a three-event trace semantics. The above proposals lack of an integrated view of diagrams or, more accurately, UML submodels of the system under development as our own proposal does (Benghazi et al., 2007).

To demonstrate the consistency between UML-RT submodels we define a partial semantic mapping into a common semantic domain given by CSP+T trace semantics. In this work we give a semantic foundation of the UML–RT composite diagrams elements and the consistency conditions required to assure that the obtained system specification fulfills the time constraints modeled in UML–RT. Our final challenge is to demonstrate the semantic consistency of a UML–RT model and its corresponding CSP+T specification.

The paper is organized as follows. In the next section we give a brief description of our method. Section 3 shows a overview on UML–RT and CSP+T, as specification languages supporting our approach. In section 4, we describe the mapping between UML–RT entities and CSP+T semantic domain to establish the semantic basis of the method. In section 5, we establish the consistency conditions to check the behavioural consistency of the models. In section 6, we apply the products of this work to the Production Cell case study. The last section gives a conclusion and discusses future work.

2 TRANSFORMATIONAL METHODOLOGY

We can perform the specification of the structural and behavioral aspects of RTS methodologically (Benghazi et al., 2007). These two different viewpoints of a system are usually attained in UML–RT by using Class and Structure diagrams, and by using SMDs, respectively. We apply a transformational method, based on a proposed set of transformation rules (Benghazi et al., 2006), which allow us to create a CSP+T model from a UML–RT analysis model of a given RTS. As it can be seen in Figure 1, the process is divided into two main phases: the first one (top–down modelling process) to model the system using UML–RT, while the second one (bottom-up specification process) obtains the formal specification in terms of CSP+T by the transformation of each UML–RT submodel.

As Figure 1 shows, mapping links are continuously established between the UML–RT diagrams of components in which the system is structured and their formal specifications in terms of CSP+T processes. These links demonstrate how CSP+T syntactical terms are used to represent the real–time constraints and the internal components and connectors that constitute the system architecture, at different levels of description detail.

In the semantic domain, lower level processes refine these of the upper levels in the system design. For instance, let us consider $P_{i:2\ldots n} = \{[\![\textit{State machine diagrams}(i)]\!]\}$, $P_1 = \{[\![\textit{Composite structure diagrams}]\!]\}$, $P_0 = \{[\![\textit{Class diagrams}]\!]\}$, where $P_0,\ P_1,\ P_i$ are structured process terms of CSP+T representing different viewpoints of UML–RT diagrams of a system. The following refinement relation between processes, $P_0 \sqsubseteq P_1 \sqsubseteq \|_{i:2\ldots n} P_i$ holds.

Process specifications may be constructed from the specifications of their components, in a way that reflects the trace semantics of CSP operators. By the application of parallel composition and hiding operators are obtained the more abstract processes within the specification phase of the system development. In (Schneider, 2000) are explained a series of rules demonstrating that the verification of the entire target system can be obtained from the composition of subsystems correction proofs, which is a result of the fact that the *satisfaction* relation is closed w.r.t. the conjunction operator,

$$\textit{If } P_i \models S_i \ \Rightarrow \textit{System} \equiv (\|_{i:1\ldots n} P_i) \models \ (\textstyle\bigwedge_{i:1\ldots n} S_i)\,;$$

thus, this result allows us to carry out the analysis of complex systems by compositional refinement.

3 THE SPECIFICATION LANGUAGE

CSP+T (Žic, 1994) is a new real–time specification language which extends CSP (Communicating Sequential Processes) (Hoare, 1985) by introducing a new set of constructs to allow the description of complex event timings from within a single sequential process, thereby providing a valuable insight into the behavioural specification of RTS.

A CSP+T process term is defined as a tuple $(\alpha P, \mathcal{P})$, where $\alpha P = \textit{Comm\_act}(P) \cup \textit{Interface}(P)$ and αP is named the *communication alphabet* of $\mathcal{P}$. These communications represent the events that process P receives from its *environment* (made up of all the other processes in the system) or those that occur internally, such as the event τ which is not externally visible.

The CSP+T language is defined by the following grammatical rules. Given a set $\mathcal{E}$ of events, a set $\mathcal{M}$ of marker variables, a set $\mathcal{I}$ of time intervals, and a set

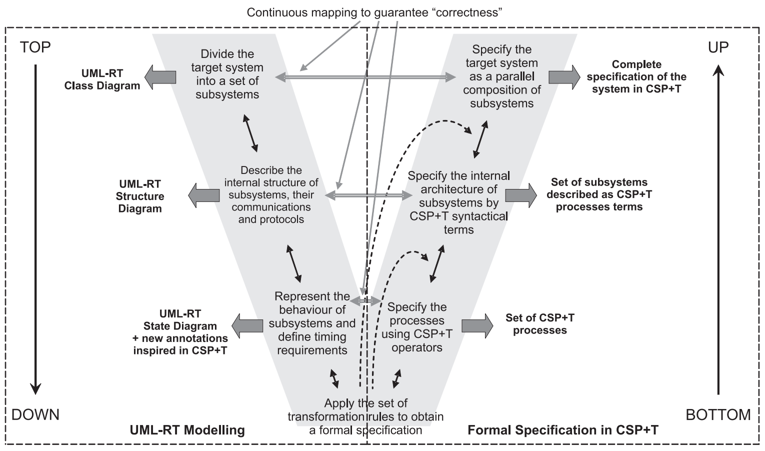
Continuous mapping to guarantee "correctness"
TOP
DOWN
UP
BOTTOM
UML-RT Class Diagram
Divide the target system into a set of subsystems
Specify the target system as a parallel composition of subsystems
Complete specification of the system in CSP+T
UML-RT Structure Diagram
Describe the internal structure of subsystems, their communications and protocols
Specify the internal architecture of subsystems by CSP+T syntactical terms
Set of subsystems described as CSP+T processes terms
UML-RT State Diagram + new annotations inspired in CSP+T
Represent the behaviour of subsystems and define timing requirements
Specify the processes using CSP+T operators
Set of CSP+T processes
Apply the set of transformation rules to obtain a formal specification
UML-RT Modelling
Formal Specification in CSP+T

Figure 1: Methodological approach to derive a correct and complete formal specification of RTS (Benghazi et al., 2007).

of process names $\mathcal{P}$, the syntax of CSP+T is given by

$$\begin{aligned} P ::= \;& Stop \mid Skip \mid e \bowtie v \to P \mid P; Q \mid P \Box Q \mid \\ & P \sqcap Q \mid P \setminus A \mid P \,|[A]|\, Q \mid P \,|||\, Q \mid \\ & I(t,T).a \to P \mid I(t,T) \to P \end{aligned}$$

where $e \in \mathcal{E}$, $A \subseteq \mathcal{E}$, $v \subseteq \mathcal{M}$, $I \subseteq \mathcal{I}$, $P \subseteq \mathcal{P}$. The interpretation of some operators is as it follows. The processes *Stop* and *Skip* represent, respectively, *deadlock* and *termination*. The prefix processes $e \bowtie v \to P$ annotates that the process first engages in the event e, then stores in the marker variable v the event occurrence of e, and then becomes the process term P. The processes $P \sqcap Q$ and $P \Box Q$ represent *internal* and *external* choice between P and Q, respectively. That means, while $P \sqcap Q$ performs an internal (τ–)action when evolving into P or into Q; on the other hand, $P \Box Q$ requires the occurrence of an observable action of either P or Q. For example, $(a \bowtie v_a \to P) \sqcap (b \bowtie v_b \to Q)$ internally chooses to become either $a \bowtie v_a \to P$ or $b \bowtie v_b \to Q$. Instead, $(a \bowtie v_a \to P) \Box (b \bowtie v_b \to Q)$ offers the communication events to its environment $a \bowtie v_a$ or $b \bowtie v_b$ and evolves into P or Q, respectively, depending of an external choice. This distinction shall be relevant for the translation of SMDs that is performed in this work. The process $P \setminus A$ behaves like P except that all occurrences of set of events A are hidden. Finally, the *parallel composition* $P \,|[A]|\, Q$ results in a *interleaving* of P and Q, which is denoted as the process term $P \,|||\, Q$, except for the events in A on which P and Q must synchronize.

CSP+T is a superset of CSP, as a major change to the latter, the traces events are now *pairs* denoted $t.e$, where t is the global *absolute* time at which event e is observed. The operators related with timing and enabling–intervals included in CSP+T are (Žic, 1994):

- The special process instantiation event denoted $\star$ (star). This event is unique in that it must be associated with a unique, global time, at which the system of processes may start.
- The time capture operator ($\bowtie$) is associated to time stamp function $a = s(e)$ that allows storing in a variable a the occurrence time of an event e when it occurs.
- The event–enabling interval can be seen from two viewpoints. When we write $I(t_1,T).a \to P$ the process P can only engage in the event a if it occurs within the time interval (t_0+t_1, t_0+t_1+T). But when we write $I(t_1,T) \to P$ the process P is executed in the instant after the "expiration" of the time interval (t_0+t_1, t_0+t_1+T), since there is no event to engage to within this interval.

4 UML–RT ENTITIES AND CSP+T

As UML establishes, the behavior of UML–RT structural elements (capsules and protocols) are specified among other UML diagrams by SMDs) A UML–SMD represents a hierarchical automaton that includes initial states, final states, choice states, simple states, composite states and transitions between states (OMG, 2004). The syntax for a transition label has three parts, all of which are optional: *Event[Guard]/Action*. A guard is a logical condition and a guarded transition may only occur if the guard is true.

The graphical representation of a CSP+T process term correspond to the simplest form of SM, hence we can map transitions of SMs to events of CSP+T and states of SMs to processes of CSP+T.

The one–to–one correspondence between states and process definitions facilitates our formal reasoning about the dynamic behaviour of processes. However, it is not enough to capture the behaviour of a SM associated to a capsule precisely. As established by UML–RT, each capsule operates according to a SM with a clear interface, which responds to events and generates actions through ports. To differentiate between external events and operations, we define roles that a port can perform. A signal arriving to a capsule port from the environment is usually interpreted as a event by a capsule SM; on the other hand, the signals exiting from a capsule are the result of executing the operations. The following definition allows specifying UML–RT capsule SMs in terms of CSP+T processes.

Definition 1 (Capsule SM). A capsule *SM* is represented in terms of CSP+T processes as a tuple $SM = (S, \mathcal{E}, \mathcal{B}, \mathcal{M}, \mathcal{I}, \mathcal{O}, \mathcal{P}, Port)$, where:

- $\mathcal{S}$ is a finite set of processes
- $\mathcal{E}$ is a finite set of events
- $\mathcal{B}$ is a set of boolean expressions (guards)
- $\mathcal{M}$ is a set of marker variables
- $\mathcal{I}$ is a set of event–enabling intervals
- $\mathcal{O}$ is a set of operations
- *Port* is a finite set of port names
- A process relation $\mathcal{P} \subseteq \mathcal{S} \times \mathcal{E} \times \mathcal{B} \times \mathcal{M} \times \mathcal{I} \times \mathcal{O} \times Port$, where $(s, e, b, a, \mathrm{I}, c, p) \in P$, is denoted by

$$\begin{aligned} P = \quad & \{b\}\mathrm{I}(T, t_1).trigger(e) \bowtie a \rightarrow (effect(c) \rightarrow P') \\ & \Box \\ & \mathrm{I}(t_1 + T) \rightarrow Timeout \end{aligned}$$

  where:
  - $a = s(e)$ corresponds to the time stamp of occurrence of the event e
  - T is the max time, started from the time t_1, in which the process P wait for event e
  - $\mathrm{I}(t_1 + T)$ represent an instant of time.
  - $port : \mathcal{E} + \mathcal{O} \rightarrow Port + \{Int\}$ is a function that associates to each event or operation x a port name in *Port* according to a distinguished label *Int* which designates the type of event or effect, where:
    - ∗ for all $e \in \mathcal{E}$, if $port(e) = Int$ then $trigger(e) = e$ else $trigger(e) = port(e)_{in}?e$.
    - ∗ for all $c \in \mathcal{O}$, if $port(c) = Int$ then $effect(c) = c$ else $effect(c) = port(c)_{out}!c$.

The above definition is semantically coherent with the one given by UML–RT w.r.t. the required behaviour associated to the capsule, that is specified according to CSP+T semantics. The capsule reacts to the received signals through its ports and sends signals through other ports too. Therefore, the definition 2 is completed by definition 3 that is actually a CSP+T extension of the CSP one in (Engels et al., 2001b).

Definition 2 (Process view). Let $SM = (\mathcal{S}, \mathcal{E}, \mathcal{B}, \mathcal{M}, \mathcal{I}, \mathcal{O}, P, port)$ be a capsule SM, let $CSP + T(SM)$ be the process associated to *SM*, and $p \in Port$. Define

$\mathcal{E}' = \{\mathrm{I}.p'_{in}?e \mid p' \neq p\} \cup \mathcal{E}_{Int}$ and

$\mathcal{O}' = \{p'_{out}!c \mid p' \neq p\} \cup \mathcal{O}_{Int}$.

Then, the process view with respect to the port p is defined as $V_p(CSP + T(SM)) = CSP + T(SM) \setminus (\mathcal{E}' \cup \mathcal{O}')$

This definition uses the hiding ($\setminus$) operator, defined in section 3. The process $CSP + T(SM) \setminus (\mathcal{E}' \cup \mathcal{O}')$ behaves like the process $CSP + T(SM)$ except that all occurrences of the events in $\mathcal{E}' \cup \mathcal{O}'$ are hidden. All these events are removed from the interface of the process, since no other process are required to engage in them, and thus the events become internal to the process $CSP + T(SM)$. The process view specifies the events that occur in the capsule SM which are of interest for a particular interaction or which are received/sent from/to a particular port. In the definition 2, $V_p(CSP + T(SM))$ represents the view of the capsule state machine $CSP + T(SM)$ from other capsule SM connected to it through the port p.

The behavioural organisation of the structural elements of UML–RT (mainly capsules) is specified through Composite Structure Diagrams. The most important aspect of this UML–RT diagram is that reflects the compositional and concurrent aspects of processes, which characterizes the dynamic behaviour of any system and becomes of paramount importance for RTS. This composition is made of connectors and ports, but *UML–RT does not set anything about the behaviour of connectors.*

UML–RT settles down that every capsule SM is associated to an event queue where input messages are stored. Connectors are responsible for carrying out the task of storing and ordering the events. We define a extension of the uni–directional connector with capacity n and non–blocking behaviour with time constraints specification, as it follows:

Definition 3 (Connector). A connector is $C = (N, In, Out, \mathcal{M}, \mathrm{I})$ where:

- N is the capacity of the buffer
- *In* is the input channel
- *Out* is the output channel
- $\mathcal{M}$ is a set of marker variables
- $\mathcal{I}$ is a set of enabling intervals
- A buffer relation $B \subseteq N \times In \times Out \times \mathcal{M} \times \mathcal{I}$, where

$(n, in, out, a, \mathrm{I}) \in B$, is denoted by

$$\begin{array}{lll} let & B(\langle\rangle) = & (I_i.in?x \bowtie a_i \rightarrow B(\langle\rangle) \\ & & \Box I_i \rightarrow Timeout) \\ & B(s) = & (I_o.out!head(s) \bowtie a_o \rightarrow B(tail(s)) \\ & & \Box I_o \rightarrow Timeout) \\ & & \Box(\#s \geq n\&I_i.in?x \bowtie a_i \rightarrow B(s) \\ & & \sqcap \#s < n\&I_i.in?x \bowtie a_i \rightarrow B(s^\frown\langle x\rangle)) \\ within & B(\langle\rangle) & \end{array}$$

where:

– $a = s(x)$ corresponds to the time at which a message contained in x is received/sent

– I_i and I_o represents the input enabling interval and the output enabling interval, respectively.

The parallel composition in CSP+T is a mechanism that puts two processes together to evolve concurrently. The interactions that occur between component processes over time (or "communications") may be regarded as events that require simultaneous participation of both processes. In UML–RT two capsules executing concurrently can be described by the CSP+T operator $|[\ ]|$ and a connector. This construct is called in (Engels et al., 2001b) "capsule–connector–capsule".

Definition 4 (Capsules composition). Let A and B be two capsules connected by two ports $p1$ and $p2$ through a connector associated to a previously defined connector process C, and let the capsule SMs associated to A and B be SM_A and SM_B, respectively. Then the semantics of this construct is defined by the following CSP+T syntactical term

$$V_{p1}(CSP+T(SM_A))\ |[p1_{in}, p1_{out}]|\ C$$
$$|[p2_{in}, p2_{out}]|\ V_{p2}(CSP+T(SM_B))$$

We denote this process as $CSP+T_{p1,p2}(SM_A, C, SM_B)$

Capsules connected through ports that intercommunicate according to previously defined protocols. In CSP+T, the protocol refers to the alphabet of communication events. The Definition 2 is a view of a capsule SM that focuses on sending and receiving messages through a particular port (Engels et al., 2001b). The alphabet of communication can be obtained by defining a relationship between the process views of capsules that communicate through a connector.

Definition 5 (Communication alphabet). Let $CSP+T_{p1,p2}(SM_A, C, SM_B)$ be the capsule composition of two capsules A and B, connected by two ports $p1$ and $p2$ through the given connector process C, and let $V_{p1}(CSP+T(SM_A))$ and $V_{p2}(CSP+T(SM_B))$ be the process views associated to SM_A and SM_B, respectively. Then the communication alphabet with respect to the connector C is defined as

$$\alpha_C(CSP+T_{p1,p2}(SM_A, C, SM_B)) =$$
$$V_{p1}(CSP+T(SM_A)) \cap V_{p2}(CSP+T(SM_B))$$

This definition allows to refine the behavioural relationship among capsules that must be respected as a contract established by the protocol implemented by the connector.

5 BEHAVIOURAL CONSISTENCY

We need to ensure the consistency from the communication and interaction viewpoint of diagrams in order to validate the UML-RT models of the target system. In this sense, we realize that the following two consistency problems may occur: (a) consistency between two capsule SMs, and (b) the consistency problem between two capsule SMs and a protocol SM. According to (Engels et al., 2003), the consistency problem (a) occurs because the interaction of two capsule SMs could lead to deadlock. Furthermore, two capsule SMs interaction must be conform to the protocol specified in the protocol SM in order to avoid the consistency problem (b). The following consistency conditions are necessary to assure the horizontal and vertical consistency (Engels et al., 2001b) according to CSP+T semantics.

Condition 1 (Deadlock freeness). Two SMs SM_A and SM_B associated to capsules A and B, and communicated by a connector with behaviour C through the ports $p1$ and $p2$, are consistent if the induced system of $CSP+T$ processes $CSP+T_{p1,p2}(SM_A, C, SM_B)$ is deadlock free.

Condition 2 (Protocol consistency). Two capsules A and B connected by a connector with behaviour C through the ports $p1$ and $p2$ are consistent with a protocol $Prot$ iff $CSP+T_{p1,p2}(SM_A, C, SM_B) \sqsubseteq_\tau Prot$.

6 CASE STUDY

6.1 The Production Cell UML–RT

The Production Cell is a standard example for evaluating methodologies for designing embedded systems (Lilius and Porres, 1999). The model includes several machines that must be coordinated in order to forge metal blanks. We decided to model one of the central element of system: the *Press*. The task of the *Press* is to forge metal blanks. Figure 2 shows its architecture.

The *Press* is composed of two basic components, the *Plant* and the *press controller (PC)*. These elements are represented by two subcapsules *Plant* and *PC* shown in Fig. 3.

The subcapsule *PC* has two ports: the port *Pp*2 to communicate with the *Press* and the port *Ppc* to

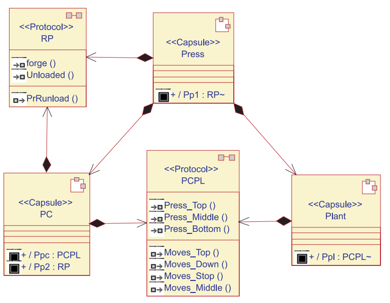

Figure 2: Press architecture – Class Diagram.

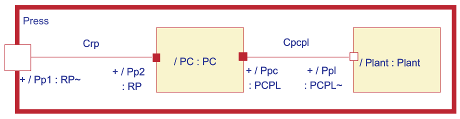

Figure 3: Press composition – Structure diagram.

receive events and to send actions to the subcapsule *Plant*. Thus, a change of state in the *PC* UML–RT SM is provoked by the reception of events sent by the *Press* or by the *Plant*. (i.e., the reception of the event *forge* from the *Press* changes the state from *Waiting_L* to *Pressing*, and the event *Press_Bottom* from the *Plant* changes the state from *Unloading* to *Waiting_U*). Figure 4 shows the SM that models the *PC* behaviour.

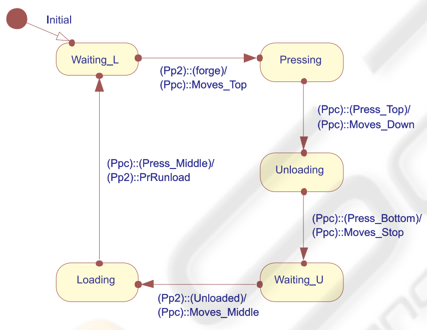

Figure 4: PC behaviour – State machine diagram.

In its turn, the *Plant* subcapsule has one port through which it receives command from the *PC* to move down, to move up or to stop, and through this port too it sends positional information when the *Plant* is placed in the bottom, middle or top position, respectively. Figure 5 shows the SM that models the *Plant* behaviour.

The original specification of the press was obtained in previous works (Benghazi et al., 2006; Benghazi et al., 2007; Capel et al., 2006) by applying a methodological approach based on a set of transformational rules.

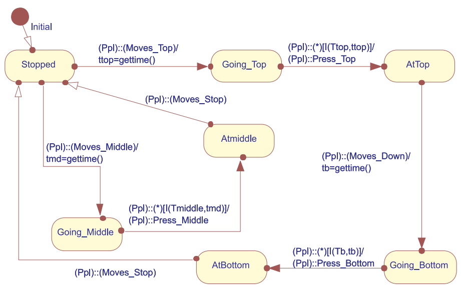

Figure 5: Plant behaviour – State machine diagram.

6.2 Checking the Behavioural Consistency

According to (Hoare, 1985), to check whether $CSP + T_{Ppc,Ppl}(SM_{PC}, Cpcpl, SM_{Plant})$ is deadlock free, it is necessary to prove that $CSP + T_{Ppc,Ppl}(SM_{PC}, Cpcpl, SM_{Plant})/s \neq Stop$

for all $s \in traces(CSP + T_{Ppc,Ppl}(SM_{PC}, Cpcpl, SM_{Plant}))$

In this sense, if we take an arbitrary trace $s \in traces(CSP + T_{Ppc,Ppl}(SM_{PC}, Cpcpl, SM_{Plant}))$, then we can see that in all the cases there is at least one event $e \in \alpha_C(CSP + T_{Ppc,Ppl}(SM_{PC}, Cpcpl, SM_{Plant}))$ whose occurrence extends the trace s; in other words, the communication between the processes SM_{PC} and SM_{Plant}, through the connector *Cpcpl* and the ports *Ppc* and *Ppl*, is established by any event of the communication alphabet in the intersection of the process views $V_{Ppc}(CSP + T(SM_{PC})$ and $V_{Ppl}(CSP + T(SM_{Plant}))$.

As regard to the second condition and according to the trace semantics (Hoare, 1985), we must check that $\tau(Cpcpl) \subseteq \tau(CSP + T_{Ppc,Ppl}(SM_{PC}, Cpcpl, SM_{Plant}))$, i.e., every trace of *Cpcpl* is also a trace of $CSP + T_{Ppc,Ppl}(SM_{PC}, Cpcpl, SM_{Plant})$. Analyzing the traces generated by the processes $CSP + T_{Ppc,Ppl}(SM_{PC}, Cpcpl, SM_{Plant})$ and *Cpcpl*, we can see that $CSP + T_{Ppc,Ppl}(SM_{PC}, Cpcpl, SM_{Plant})$ is a refinement of the process *Cpcpl*; in other words, $CSP + T_{Ppc,Ppl}(SM_{PC}, Cpcpl, SM_{Plant})$ is a more detailed description than the one represented by *Cpcpl*; thus, $CSP + T_{Ppc,Ppl}(SM_{PC}, Cpcpl, SM_{Plant})$ constitutes a specification closer to the intended system implementation. This result checks the condition 2.

7 CONCLUSIONS

In this paper is presented a complete semantic specification of UML–RT analysis entities in terms of CSP+T processes. Also, the consistency conditions of the model obtained by applying the transformational methodology are established.

Our methodological approach, introduced in a previous paper, is complemented in this work with the integration of a complete semantic set of the UML–RT analysis entities and the consistency conditions required to assure that the system obtained fulfills the time constraints modeled in UML–RT. This new contribution make more complete our method since we systematically obtain a detailed system specification from an initial UML model of high level of abstraction and the software engineer can verify his/her work at any time across the development process.

The future and ongoing work is aimed at the application of the methodological approach in other RTS cases, researching in–depth about the verification of these specifications, and the integration of our methodological approach with tools as FDR.

REFERENCES

Benghazi, K., Capel, M., and Holgado, J. (2006). Design of real-time systems by systematic transformation of UML-RT models into simple timed process algebra system specifications. *Proc. 8th International Conference on Enterprise Information Systems (ICEIS 2006)*, pages 290–297.

Benghazi, K., Capel, M., Holgado, J., and Mendoza, L. E. (2007). A methodological approach to the formal specification of real-time systems by transformation of UML-RT design models. *Science of Computer Programming*, 65(1):41–56. Special Issue Methods of Software Design: Techniques and Applications.

Bolognesi, T. and Brinksma, E. (1987). Introduction to the iso specification language lotos. *Comput. Netw. ISDN Syst.*, 14(1):25–59.

Capel, M., Mendoza, L., Benghazi, K., and Holgado, J. (2006). A semantic formalization of UML–RT models with CSP+T processes applicable to real-time systems verification. *Actas XV Jornadas de Ingeniería del Software y Bases de Datos (JISBD 2006)*, 1:283–292.

Cheng, B. and Wang, E. (2002). Formalizing and integrating the dynamic model for object-oriented modeling. *IEEE Trans. Softw. Eng.*, 28(8):747–762.

Engels, G., Heckel, R., and Küster, J. (2001a). *Rule-Based Specification of Behavioral Consistency Based on the UML Meta-model, Lecture Notes in Computer Science 2185: UML 2001*, pages 272–286. Springer-Verlag, Berlin.

Engels, G., Küster, J., and Groenewegen, L. (2003). Consistent interaction of software components. *Transactions of the SDPS: Journal of Integrated Design & Process Science*, 6(4):2–22.

Engels, G., Küster, J., Heckel, R., and Groenewegen, L. (2001b). A methodology for specifying and analyzing consistency of object-oriented behavioral models. *Proc. 8th European Software Engineering Conference*, pages 186–195.

Evans, A., France, R., Lano, K., and Rumpe, B. (1998). Developing the UML as a formal modelling notation. *Proc. UML'98 - Beyond the Notation*, pages 297–307.

Haugen, O., Husa, K., Runde, R., and Stolen, K. (2005). STAIRS towards formal design with sequence diagrams. *Software & System Modelling*, 00:1–13.

Hoare, C. (1985). *Communicating Sequential Processes*. International Series in Computer Science. Prentice-Hall International Ltd., Hertfordshire UK.

Kim, S.-K. and Carrington, D. (1999). *Formalizing the UML Class Diagram Using Object-Z, Lecture Notes in Computer Science 1723: UML 99 - The Unified Modeling Language: Beyond the Standard*, pages 83–98. Springer-Verlag, Berlin.

Lilius, J. and Porres, I. (1999). *The Production Cell: An Exercise in the Formal Verification of a UML model*, volume 288 of *TUCS Technical Report*. Turku Centre for Computer Science, Finland.

OMG (2004). *UML Superstructure Specification - version 2.0*. Object Management Group, Massachusetts, USA.

Schneider, S. (2000). *Concurrent and Real-time Systems - The CSP Approach*. John Wiley & Sons, Ltd., Chichester, England.

Žic, J. (1994). Time-constrained buffer specifications in CSP+T and Timed CSP. *ACM Transaction on Programming Languages and Systems*, 16(6):1661–1674.