Combining the Description Features of UML-RT and CSP+T Specifications Applied to a Complete Design of Real-Time Systems

M.I. Capel Tuñón, K. Benghazi Akhlaki, J.A. Holgado Terriza

Abstract—UML is a collection of notations for capturing a software system specification. These notations have a specific syntax defined by the Object Management Group (OMG), but many of their constructs only present informal semantics. They are primarily graphical, with textual annotation. The inadequacies of standard UML as a vehicle for the complete specification and implementation of real-time embedded systems has led to a variety of competing and complementary proposals. The Real-Time UML profile (UML-RT), developed and standardized by OMG, defines a unified framework to express time, scheduling and performance aspects of a real-time system. This paper presents a methodological approach for specifying to real-time systems. Therefore, we combine two methods, a semi-formal one, UML-RT, which enables the visual modelling of a real-time system, and a formal one, CSP+T, based on the CSP process algebra, which is a formal specification language that enables the specification of real-time requirements. To show the applicability of the approach, the correct design of a real-time system with hard real time constraints is obtained by applying the set of mapping rules proposed here.

Keywords—CSP+T, formal software specification, process algebras, real-time systems, Unified Modelling Language.

I. INTRODUCTION

UML-RT (Unified Modelling Language for Real-Time) [1], [2] is an industrial standard used to model real-time systems, but its modelling entities and syntactic constructions lack a defined syntax and a precise semantics. In order to unambiguously specify the behaviour of a reactive system, we have established a mapping that gives a defined meaning to UML-RT entities according to the semantic domain discussed in the following text. This study is aimed at obtaining a systematic, formally oriented analysis and design method, which also includes temporal specifications. A set of rules [3] allows us to systematically derive a verifiable design of a real-time system from a system requirements model, which includes the functional system requirements as well as the behavioural and temporal ones, which is initially constructed using UML-RT.

The formal specification language CSP+T [4], derived from the process algebra CSP [5], [6], is a good candidate to give a precise semantics to UML-RT analysis entities. UML-RT provides a visual system description of each object within the analysis model as well as its behaviour, which is diagrammatically represented by a class of state diagrams (SD) similar to Statecharts [7] ones, with all aspects concerning system functionality, behaviour and some key temporal properties of any real-time target system being specified by means of the profile given in [2]. The specification of the system’s structural aspects is carried out by constructing class diagrams as in UML, which also describe system composition and the associations among the objects.

CSP+T complements the dynamic description of the system under development (target system) by introducing timed events, which are used to define timing constraints on the execution of parallel processes, in which modern real-time systems are structured. Following the work carried out by different authors in [3], [8]-[11] regarding the translation of UML SD into correct process terms according to the CSP specification language syntax, we propose a new set of mapping rules from UML-RT entities into CSP+T syntactical terms, so as to allow the specification of complex real-time requirements.

To show its applicability, we have used the proposed method to carry out the complete specification including real-time constraints of a basic component of the “Production Cell”, which is a well known manufacturing-industry paradigmatic case. The rest of this paper is structured as follows: section 2 provides an overview of the UML-RT and the UML diagrams used in our approach and section 3 explains the system specification method proposed. Section 4 presents the specification of key Production Cell components, namely, robot arm software-controllers. Finally, some conclusions are drawn and the references are listed.
II. UML FOR REAL-TIME SYSTEMS

UML-RT extends the basic UML [12] analysis entities with constructs to facilitate the design of complex embedded real-time software systems [13]. These constructs are derived from the set of concepts initially defined in the ROOM modelling language [14]. This language focuses primarily on the specification of the architecture of real-time software systems, i.e., it identifies the major components of a target system, their externally visible properties, and the communication between them. The importance of the software architecture definition in the development cycle lies in the fact that the decisions made during the architectural design have a very significant impact on the later system design; this phase also being the one which can benefit most from a good modelling language [15].

A. Basic Concepts

UML-RT adds four new building blocks to the standard UML meta-model, three of these (capsules, ports and connectors) being used to model the structure of the system, while the fourth (protocols) models the communications within the system. The behaviour of system components is modelled using UML SD. These diagrams, which are analogous to Statecharts diagrams [7] except for minor differences, contain state variables and describe the changes to the states by syntactic expressions in a programming language. Capsules are the central modelling constructs in UML-RT. They represent the major architectural components of complex real-time systems. Another diagram describes the collaboration among capsules, so as to provide a model of the software architecture of the system. A capsule may have one or more ports through which it can communicate with other capsules and it may contain one or more collaborating sub-capsules. The behaviour of a sub-capsule may be described by a UML SD. Ports are boundary objects (usually drawn on the frontier of capsules) that are “owned” by one capsule and that provide the only way by which capsules interact with the rest of the system. Ports can be connected to either a UML SD, which defines the functionality of an, as yet, incompletely designed component of the target system, or to a sub-capsule port. Therefore, a message sent to a port can be handled directly by the capsule, or forwarded to a suitable sub-capsule. Fig. 1 shows an example of a simple UML-RT component architecture that includes these concepts.

![Fig. 1 An example of UML-RT concepts](image)

The port \( p1 \) of the capsule \( CapsA \) is connected to a UML SD, and \( p2 \) is connected to the sub-capsule \( CapsB \). In addition, the capsule has an internal port \( p3 \), the only one which is visible inside \( CapsA \), which connects a port of the sub-capsule \( CapsB \) to a UML SD.

Protocols define a number of participating roles assumed when capsules inter-communicate and also define the signals sent and received while they are engaged in a role. The definition of valid sequences of signals, which are encoded as in UML SD, need to be included in protocols as well, or otherwise any signal sequence may be considered valid.

Connectors are used to model communication channels between two or more ports, which can therefore realize different roles when they communicate using their mutual protocol. Protocols and connectors define the behaviour of the system at the architectural level.

Apart from the standard UML analysis entities, UML-RT provides additional support when modelling the architecture of interactive systems, although it does not provide a complete support for modelling temporal dependencies in real-time processes at the moment. Only timeouts are allowed as the basic structures to introduce timing constraints in a UML-RT model of a target real-time system, but these constructs do not have a precise semantics defined by the modelling language itself. Consequently, UML-RT does not facilitate carrying out temporal requirements analysis of a real-time system specification. In [15], the definition of UML-RT differs from that in [2], in that a formal semantics is given to its entities, with the structural and behavioural parts of UML-RT model being semantically defined in terms of flow graphs. This approach could be considered close to ours, even though concurrency issues are not addressed in the former, as our method does in fact by using the parallel process composition constructs of the formal specification language CSP+T.

B. UML Profile for Schedulability, Performance and Time

In 1999, OMG issued a request for proposals regarding the new UML profile [16] intended to address the specific issues that arise in the development of real-time systems. The main aim of the profile is to enable the construction of models that can be used to make quantitative predictions concerning schedulability, performance and time issues of a real-time system, to facilitate the communication of preliminary designs between developers in a standard way and to enable interoperability among various analysis and design tools.

C. Modelling Time

The infrastructure needed to support real-time system modelling is provided by the definition of two basic types of timing mechanisms, namely, timers and clocks, in the UML-RT profile. These timing mechanisms are always associated with a reference clock that provides the (simulated) time and they also have a number of attributes in common such as resolution and drift. Timers are mechanisms that may generate a timeout event when a specified time instant occurs. Timeout events may occur either when a clock reaches a pre-defined
value or when a pre-specified time interval expires in relation to a given instant (usually the instant when the timer was started). Clocks are mechanisms that periodically cause a clock tick event to occur.

D. Modelling Schedulability

The UML-RT profile describes a set of common scheduling annotations, which are sufficient to enable the basic schedulability analysis of real-time tasks to be performed. Individual tool vendors are expected to provide specialized annotations to allow for more extensive analysis in the future. The annotations defined by the present profile include priority, absolute and relative deadlines, and worst-case completion time.

E. Modelling Performance

OMG describes a profile [2] that is intended to provide general facilities for capturing performance requirements within the design phase of the development cycle. A set of performance specialized stereotypes and their association constraints are defined by the association of performance and QoS (Quality of Service) indicators with selected elements of a UML model; each stereotype may have one or more tags to denote its corresponding performance values.

III. CSP WITH COMPLEX EVENT TIMINGS : CSP+T

CSP+T [3] is a new real-time formal specification language, which adds expressive power to some of the sequential aspects of CSP and allows the description of complex event timings from within a single sequential process, thereby providing valuable insights into the behavioural specification of real-time processes.

In the group of CSP derivatives to describe time intervals, we should mention Timed CSP [4]-[6] and CSP+T, the latter being a simpler approach although still powerful enough to formally describe a set of deterministic processes with time constrained behaviour. CSP+T is a formal specification language that is adequate for the majority of real-time systems. The syntax of CSP+T, which is a superset of the CSP one, has been adapted to our method. The differences between the two formal specification languages are described as follows:

- Every process \( P \) defines its own set of communication symbols, termed the communication alphabet \( \alpha(P) \). These communications represent the events that process \( P \) receives from its environment (constituted of all the other processes in the system) or that occur internally, such as the event \( r \) which is not externally visible. External events can be understood as the pure synchronization between an asynchronous process and its environment. Any type of event causes a state change of the process in which it is observed.

- The communication interface \( \text{comm}_-\text{act}(P) \) of a given process \( P \) contains all the CSP-like communications [6], i.e. the synchronous, one-to-one, communications between parallel processes, in which process \( P \) can engage and it also includes the alphabet \( \alpha(P) \) representing signals and events occurring in \( P \). Therefore, the communications of process \( P \) are given by the set \( \text{Comm}_-\text{act}(P) = (\text{Interface}(P) \cup \alpha(P)) \).

- A new operator, * (star), is introduced in the programming notation to denote process instantiation. An instance of a process term must be created before it can execute. This event is unique in the system since it represents the origin of a global time at which processes can start their execution. As an example, let us consider a process \( P \) that initially can only engage in the event \( a \). In CSP, this process, would be denoted as: \( P = a \rightarrow \text{STOP} \), but it must be instantiated before being executed in CSP+T. Given \( P' \), the timed version of \( P \), which is instantiated at time 1, where \( s \) is a time stamp associated to the abstract communication \( a \), the specification of \( P' \) becomes,

\[
P' = 1. * \rightarrow s.a \rightarrow \text{STOP}, \text{ where } s \in [1, \infty].
\]

It should be noted that event \( a \) occurs only once in the interval.

- A new event operator \( \geq \) is introduced to be used jointly with a “marker variable” to record the time instant at which the event occurs. \( ev > v \) means that the time at which \( ev \) is observed during a process execution is in the marker variable \( v \). The value of time stamps is taken from the set of positive real numbers, so that successive events form a non-decreasing monotonic sequence. As several successive events can instantiate the same variable at different times, if we specify the process \( P \) as follows

\[
P = 1. * \rightarrow a \geq \text{var} \rightarrow \text{STOP},
\]

for each process execution, the marker variable \( \text{var} \) will record the corresponding time value at which event \( a \) occurred, and it will always satisfy \( \text{var} > 1 \). The scope of marker variables is strictly limited to one sequential process. They cannot be referenced or accessed in any other way within a concurrent composition of processes.

- Each marker event is usually associated with a time interval, which is called its “event-enabling” interval and represents the period of time over which the event is continuously available to the process and its environment. The initial times for intervals are relative to a preceding event or to a marker variable, which is instantiated during current process execution. A process is considered to be the STOP process if it cannot engage in the marker event or in an alternative event during the enabling interval. Let us suppose, for instance, that there is a process \( P \), a process which can only engage in event \( a \), which can only occur between 1 and 2 units of time from the process instantiation time (the preceding event), recording in the marker variable \( \text{var} \) the time at which the event \( a \) occurred. The specification of this process is therefore,

\[
P = 0. * \rightarrow [1, 2].a \geq \text{var} \rightarrow \text{STOP}
\]
After the process execution, the value of the marker variable satisfies the inequality $1 \leq v \leq 2$.

The enabling interval can be defined in a more compact way by using the function $I$, $I(T, v)$, where $v$ is the marker variable that records the time instant at which the preceding event occurred, and $T$ defines the duration of the time interval starting at the time instant stored in $v$. An example is:

$$P = 1 \cdot \forall a : \left(a \prec v \rightarrow I(3, v).c \rightarrow d \rightarrow \text{STOP}\right)$$

in which the event $c$ can occur at least three time units after the process $P$ engages in the event $a$.

If the marker variable does not appear in the signature of function $I$, the enabling interval is relative to the previous marker variable in the scope of the process, otherwise the enabling interval for that process is considered the default interval $[0, \infty]$. The times for events are absolute and the times for intervals are relative to the preceding time stored in marker variable.

- The semantics of the parallel composition of two processes with enabling intervals which must be synchronized depends on whether the values of these intervals are identical, partially overlapping or disjoint. In the first case, the processes synchronize on the common initial events, as established in overlapping or disjoint. In the first case, the processes synchronize on the common initial events, as established in CSP communication semantics, i.e., given $P = E_1.Q$ and $R = E_2.S$, then

$$P/Q \neq \text{STOP} \quad \text{iff} \quad a(Q) \cap a(S) \neq \emptyset \land E_1 \cap E_2 \neq \emptyset.$$  

In the case of disjoint enabling intervals ($E_1 \cap E_2 = \emptyset$), the parallel composition of processes behaves as the STOP process.

IV. FORMAL SPECIFICATION AND TRANSFORMATION METHODOLOGY FROM UML/RT

Because real-time systems verification is mostly dependent on the previous correct specification of temporal and dynamic system properties, we are particularly interested in systematically obtaining a complete and accurate specification of the timing constraints, behavioural aspects and software architecture of real-time systems.

The multi-functionality of real-time systems, the concurrency of their active objects and their temporal requirements make them complicated to model. To manage the complexity of this kind of systems, a bottom-up, semi-formal, compositional strategy is chosen. This method combines two specification notations, UML-RT and CSP+T. The methodological procedure consists of first dividing the system into a set of subsystems and then considering the architecture of the entire system model (Fig. 2) as a net of capsules (components) connected by ports (connectors) that intercommunicate according to a previously defined protocol, which can be specified separately. The description of the dynamic aspects of a subsystem is obtained by specifying its behaviour using UML SD, which is afterwards transformed into a CSP+T specification. The class diagram illustrates the architecture of software components and the dependencies among them. The real-time system specification obtained, which is the combination of a UML SD diagram and a class diagram, gives a global view of the target system, including its dynamic, static and timing aspects.

Nevertheless, UML lacks a precise semantics and it does not provide any means of defining explicit temporal constraints or properties, other than the temporal ordering of events, and thus our contribution consists in combining an initial UML real-time system model with a CSP+T system specification, which adds expressive power to some of the sequential aspects of the initial model as well as enabling the description of complex timing constraints and temporal dependencies among real-time processes.

CSP+T can be considered a formal specification language to be used in the design of the majority of real-time systems. The combination of UML-RT and CSP+T features is carried out by the application of a series of transformation rules, which allow us to create a complete system specification, including behavioural and timing properties, in CSP+T, from an initial UML-RT model of a real-time system. These transformation rules, discussed in [3] for a different modelling language, are applied starting from the active objects in class diagrams, whose behaviour is described by a UML SD, which are transformed to CSP+T processes. We follow a procedure consisting of the following steps:

1. First, we define the dynamic behaviour of all components in the system using UML SD, and then, for all the active objects, we define:
   a. Initial State, the starting point of the system;
   b. All the states which an object passes through;
   c. For all events and actions triggering state transitions of objects, perform the following steps:
      i. Find the marker events and the restricted ones;
      ii. Assign a special function `get_time()` to the marker event, so the occurrence instant is obtained;
      iii. Assign an enabling interval to the restricted event;
      d. Identify all the transitions triggered by a special timeout event, which serves to model the situation in which a restricted event $e_2$ does not occur within the enabling interval. See rule 3 of Table I as an example of...
this scenario.

2. Transform each UML SD diagram into a CSP+T process:
   a. Map each state into a CSP+T process, the initial state being assigned to a process term that includes the instantiation event (rule 1), which gives the global time origin;
   b. Any transition from \( P \) to \( Q \) triggered by a marker event \( e \) is translated into the CSP+T process \( P \xrightarrow{e} Q \) where \( te \) is the instant of the event occurrence. This mapping is summarized as rule (2);
   c. There are two possible representations of choices: a choice state (represented as a diamond shape) or a normal state with more than one outgoing transition. In the choice state, the decision on which branch to take next depends on the prior actions performed by the process within the same execution step. In a normal state, the choice depends on the trigger event that occurs upon exiting from the current state (rule 4).

3. Create a class diagram for modelling the whole system to show the relations between the system components:
   a. Model all system components (subsystems) as capsules;
   b. Model the interaction between capsules as protocols;
   c. Capsule operations are private and protocol operations are public.

4. To combine the individual processes obtained in step 2, we transform the system class diagram into CSP+T processes:
   a. Treat each capsule as a CSP+T process;
   b. Capsule operations become process internal events of the processes;
   c. Protocol operation denotes the communication between two capsules, otherwise the signals that are shared between two processes;
   d. Two associated capsules are presented as two processes composed in parallel with all the events in their common protocol hidden (rule 5);
   e. Processes associated to the classes are progressively composed in parallel and the operations appearing in the associated protocol become hidden (rule 6);
   f. The transformation finishes when all the classes are composed and all internal events (private operations) are hidden.

So far, the above method has only been applied to the bottom-up specification of a set of deterministic processes with time constrained behaviour, due to the fact that the real-time systems addressed to date have mostly been device controllers, which are deterministic pieces of software, because their predictability as software components is usually preferred to their programmability. Nevertheless, we are now successfully applying the method to non-deterministic processes in the development of more complex software controllers.

A. Example
Let us examine a simple example to illustrate these steps (4.e-f):

CapsA interacts with CapsB within a protocol ProA-B, and with CapsC within a protocol ProA-C, as shown in Fig.3.

![Fig.3 Connection diagram of UML-RT capsules](image)

Let \( Sys \) be the process that models the system composed of two capsules, CapsA and CapsB:

\[
Sys = \{\text{CapsA} \parallel \text{CapsB}\} \setminus \{\text{Events Prot A-B}\},
\]

and let \( Sys_1 \) be the process that models the system composed of \( Sys \) and CapsC:

\[
Sys_1 = \{\text{Sys} \parallel \text{CapsC}\} \setminus \{\text{Events Prot A-C}\}.
\]

Therefore, the method is compositional, i.e. the two subsystems represented as processes are encapsulated in the process term \( Sys_1 \).

B. Timing Constraints
For two successive marker events \( (e_1, e_2) \), we assign to \( e_1 \) (the preceding event) a marker variable \( v \) to record the time at which the event occurs and to \( e_2 \) (the successor event) an enabling interval \( (T, v) \), with \( T \) being a time interval which takes enough time so that the event \( e_2 \) can occur (rules 3 and 4), meaning that the occurrence of \( e_2 \) is restricted to the time \( T \) from the occurrence of event \( e_1 \); otherwise, if the event does not occur within the enabling interval \( I \), a special event timeout is triggered to bring the system state to a null state (\( \text{SKIP} \)). The complete definition of marker events, variables and enabling intervals can be seen in [3].
### TABLE I
**MAPPING RULES FROM UML/RT TO CSP+T**

<table>
<thead>
<tr>
<th></th>
<th>StateChart Diagram + Class Diagram</th>
<th>Description</th>
<th>CSP+T Model</th>
</tr>
</thead>
</table>
| 1. | ![Initial State Diagram](image) | Initial State | Sys = 0. * → A  
(*: instantiation event) |
| 2.1 | ![Transition from Simple State A to B](image) | Transition from a simple State A to a simple State B triggered by a marker event e | A = e >< m_e → B |
| 2.2 | ![Transition from Simple State A to Composite State with Initial State Bi](image) | Transition from a simple State A to a Composite State with an initial State Bi | A = e >< m_e → Bi |
| 2.3 | ![Transition from Composite State with Final State Af to Simple State B](image) | Transition from a Composite State with a final State Af to a Simple State B | A_f = e_f → e → B  
A_f is a final state in a composite state |
| 3. | ![Two Successive Events](image) | (e_1, e_2) two successive events, e_1 is a marker event and e_2 is its restricted event | A = e_1 >< m_e_1 → B  
B = (((T, m_e_1), e_2 → C) | I(T, m_e_1) → Timeout → Skip). with T ∈ R^+ |
| 4.1 | ![External Choice](image) | The choice of which branch to take depends on the trigger event occurring upon exiting from the current state | A = (e_1 & b_1 → B □ e_2 & b_2 → C)  
If (e_1 ≠ e_2) we can write :  
A = (e_1 & b_1 → B | e_2 & b_2 → C)  
Operator □ represents non-deterministic and operator | represents deterministic choice.  
A = (\([0, T_1], e_1 → B\) n \([T_1, T_2], e_2 → C\)) with 0 < T_1 < T_2 |
| 4.2 | ![Internal Choice](image) | The decision on which branch to take depends on the prior action within the same execution step | A = (\([0, T_1], e_1 → B\) n \([T_1, T_2], e_2 → C\)) with 0 < T_1 < T_2 |
| 5. | ![Association between two capsules sharing a protocol](image) | Association between two capsules sharing a protocol | Sys = \{A/B\} \{E_p\}  
Ep: a set of protocol operations |
| 6. | ![Association between more than two capsules](image) | Association between more than two capsules | Sys = \{A/B\} \{E_{AB}\}  
The protocol common to capsules A and B is hidden from the environment  
Sys1 = \{Sys/C\} \{E_{AC}\} |
C. Modelling Class Diagram
- Modelling Protocol: Each two capsules associated within a class diagram exchange a sequence of signals defined in protocol \( Pt(CN, Ep) \), with \( CN \) being a pair of the form \((c_1, c_2)\) in which \( c_1, c_2 \in CS \) (set of capsules) and \( c_1 \neq c_2 \), and where \( Ep \) is a set of events shared between two capsules.
- Modelling Associations in Class Diagrams: an association in a class diagram is modelled as a parallel composition in \( CSP+T \) made up of two capsules, with the events of its associated protocol having been turned into hidden events.

V. THE PRODUCTION CELL CASE STUDY

The case study [17] presents a realistic industry-oriented problem, where safety requirements play a significant role, and which can be addressed by the application of formal methods. The manageable size of the Production Cell (PC) design task allows us to experiment with various approaches (Fig.4).

The PC processes metal blanks which are conveyed to a press by a feed belt. A robot arm takes each blank from the feed belt and places it on the press, the robot arm then withdraws from the proximity, after which the press processes the metal blank and opens again. Finally, another robot arm takes the forged metal plate out of the press and puts it on a deposit belt (see Fig.5).

The PC processes metal blanks which are conveyed to a press by a feed belt. A robot arm takes each blank from the feed belt and places it on the press, the robot arm then withdraws from the proximity, after which the press processes the metal blank and opens again. Finally, another robot arm takes the forged metal plate out of the press and puts it on a deposit belt.

A normal work cycle of the robot can be described in four main steps:
1. The robot rotates clockwise until Arm 1 is faced to the table, when it extends and picks up the metal blank on top of it.
2. The robot rotates counter clockwise until Arm 2 points towards the press, when it extends and picks up a forged piece.
3. The robot rotates counter clockwise until Arm 2 points towards the deposit belt, when it extends and drops the piece onto the belt.

Fig.4 (a) Production Cell configuration schema

Fig.4 (b) View of the Physic model

Fig.5 Robot and press (top view)

Fig.6 The Robot component class
4. The robot rotates counter clockwise until Arm 1 points towards the press, when it extends and places the blank onto it.

The Robot Class Diagram (Fig.6) shows the robot software architecture, i.e. the interaction between the robot controller and the two arms of the robot, by defining its classes and their associations, but it does not describe the behaviour of the class instances. Subsequently and still within the specification process, UML SD normally is used to model the behaviour of the robot controller and the two robot arms.

The proposed method, by applying some of the mapping rules in Table I to a Robot UML SD diagram, helps the analyst to obtain an interactive, temporal specification of the rules in Table I to a Robot UML SD diagram, helps the analyst to obtain an interactive, temporal specification of the robot-controller, specified as a CSP+T process term. The processes Robot Controller and Arm1 (which represents the capsule that hides the robot arm hardware) are composed in parallel, hiding the protocol operations in PArm1 (the protocol with the signals shared between the two capsules):

To avoid collisions between the arms and the other PC components (press, belts, etc.), some of which have to be loaded or unloaded, we store in a variable tposx the time at which the robot arrives at a given position in each composite state of the robot. An interval I(TCL/U, tposx) is assigned to the event which warns the controller that the component is ready to be loaded or unloaded by the robot arm. The arm can extend only if the event occurs within the enabling interval, otherwise the timeout event is triggered and the robot exits the current state and turns towards another position to complete its task. To allow safe rotation the arm must be retracted before the robot can turn. The robot UML SD in Fig.7 shows the integration of these concepts, which is needed to describe the PC safety requirements.

![Fig. 7 Robot UML SD Diagram](image-url)

RobotController-Arm1 = (Robot controller // Arm1) \ {A1Extend, A1Retract, A1Load, A1Unload, A1Stop}

Arm1 with Arm2 we obtain the Robot process structure:

Robot = (Robotcontroller-Arm1 // Arm2) \ {A2Extend, A2Retract, A2Load, A2Unload, A2Stop}

By composing in parallel the processes RobotController-
Applying the mapping rules from UML-RT to CSP+T to a Robot SD Diagram we obtain:

Robot-Controller = RC
RC = Start → CW
CW = Pos1 ¬< tpos1 → WFT
WFT = ((I (T_DB, t_pos2).TableReady ¬< t_pos3 → A1.extend) → WA1E1 = I (T_TR, t_pos1).TableReady ≫ t_pos1 → Turn (left) → CW
WA1E1 = I (T_TR, t_pos1).TableReady ≫ t_pos1 → Turn (left) → CW
WA1E1 = I (T_TR, t_pos1).TableReady ≫ t_pos1 → Turn (left) → CW
WA1E1 = I (T_TR, t_pos1).TableReady ≫ t_pos1 → Turn (left) → CW

We recursively compose the translated CSP+T terms in parallel, which correspond to the classes of the diagram, and hide the messages of the protocol that are only used to connect two capsules at each composition step:

Robot-Press = (Robot // Press \ {PressReadyLoad, PressReady Unloaded, forge})
DB-Robot-Press = (Robot-Press // DB \ {Place, DBEmpty})
Tabe-DB-Robot-Press = (DB-Robot-Press // table) \ {TableReady, Unloaded})
FB-Table-DB-Robot-Press = (Table-DB-Robot-Press // FB) \ {FBReadyLoad, Loaded, Unloaded})

The bottom-up design is completed by defining the following instantiated CSP + T process that models the whole system and which is derived from the previous FB- Table-DB-Robot-Press term:

Production_Cell_Context = PCC
PCC = 0. 0 → (FB- Table-DB-Robot-Press) \ {GetPosition, Turn(Left) Turn(Right) Turn(Stop), Engine, PressTop, PressMiddle, PressLower TableMove, tableTurn, TableStop)

VI. CONCLUSION

In this paper, we describe a systematic method to derive the correct system specification of the “Production Cell”, starting from a semi-formal system user requirements specification model in UML-RT. Our approach combines UML-RT with CSP+T to overcome the imprecision that UML models present in describing real-time systems. The future and ongoing work in our project is aimed at using the proposed method for automatic code generation of embedded control real-time systems and at attaining integration and interoperability with state-of-the-art UML-RT software tools, such as the ObjectTime [12] one.

ACKNOWLEDGMENT

The authors thank Anna Gimán Padua (Departamento de Procesos y Sistemas, Universidad Simón Bolívar, Caracas, Venezuela) for her help with the first draft of this work.

REFERENCES


**Manuel I. Capel Tuñón** is Professor at the Software Engineering Department, University of Granada, where he studied Physics and obtained his PhD in Computer Science in 1992. He is now leading a research group in the field of Concurrent Systems. His fields of interest are related to the application of formal methods to the specification of reactive systems, such as Temporal Logic and Process Algebras and the development of distributed and real-time systems software using object-oriented programming paradigms and methodologies. [http://lsi.ugr.es/~mcapel](http://lsi.ugr.es/~mcapel).

**Kawtar Benghazi Akhliki** is a PhD candidate in the Software Engineering Department, University of Granada, where she currently holds a grant from the AECI (Agencia Española de Cooperación Internacional).

**Juan A. Holgado Terriza** received the PhD in Physics in 2002 from the University of Granada. He is Assistant Professor at the Software Engineering Department, University of Granada. His fields of interest are related to the development of automatic tools for the specification and design of real-time and hybrid systems, distributed and concurrent object-oriented programming and java programming of mobile embedded devices. [http://lsi.ugr.es/~jholgado](http://lsi.ugr.es/~jholgado).