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

Kawtar Benghazi Akhlaki, and Manuel I. Capel-Tuñón

***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 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 the time, scheduling and performance aspects of a system. We present in this paper a framework approach aimed at deriving a complete specification of a real-time system. Therefore, we combine two methods, a semi-formal one, UML-RT, which allows the visual modeling of a real-time system and a formal one, CSP+T, which is a design language including the specification of real-time requirements. As to show the applicability of the approach, a correct design of a real-time system with hard real time constraints by applying a set of mapping rules is obtained.

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

I. Introduction

UML-RT [1] is an industrial standard used to model real-time systems but its modeling entities and syntactic constructions lack of a defined syntax and a precise semantics. In order to unambiguously specify the behavior of a reactive system, we have established a mapping that gives to UML-RT entities a defined meaning according to the semantic domain discussed in the following text. This work is aimed at obtaining a systematic, formal oriented, analysis and design method, which also includes temporal specifications. A set of rules allows us to systematically deriving a verifiable design of a real-time system from a semi-formal system requirements UML-RT model.

CSP+T [2], which is an extension of CSP [3], [4], is a good candidate to give a precise semantics to UML-RT analysis entities. On one part, 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 Statecharts diagram [5]. On the other part, CSP+T complements the dynamic description of the target system by introducing timed events, which are used to specify timing constraints on events and actions which occur during the execution of processes in real-time systems. Following the work [6]-[8] already done to translate state diagrams into CSP process terms, we can establish a new set of mapping rules between UML-RT entities and CSP+T terms and operators in order to allow the specification of complex temporal dependences between real-time processes.

System structural aspects are specified by constructing *class diagrams* [9], which describe the system composition and associations between its objects. Thereby, we can obtain a specification of all aspects regarding functionality, behaviour and some key temporal properties of any real-time system under development.

As to show the applicability of the method, we have used it to carry out the software development of a basic component of a manufacturing industry paradigmatic case: the "Production Cell". The rest of this paper is structured as follows: section 2 provides an overview on UML-RT and the UML diagrams used in our approach, section 3 explains the system specification method that we propose here. In section 4, using the example of the Production Cell, we present a complete system specification of one of its robot arm controllers. The article ends up with some conclusions and references.

II. UML for Real-Time Systems

UML-RT extends the basic UML [9] analysis entities with constructs to facilitate the design of complex embedded real-time software systems [10]. The origin of the UML-RT modeling notation is the real-time specific modeling language ROOM [11], which has been modified to follow the UML standardized framework. The language focuses primarily on the specification of the architecture of software systems, i.e., their major components, the externally visible properties of these, and the communication between them. The importance of the software architecture definition in the development cycle is argued by considering that decisions made during the architectural design will have a very important impact on the later system design, being also this phase which can profit the most from a good modeling language.

Manuscript received received July 15, 2005. This work is supported in part by the Spanish Ministry of Education and Science under Grant MAT2004-06872-C03-03.

Kawtar Benghazi Akhlaki is a PhD candidate in the *Departamento de Lenguajes y Sistemas Informáticos*. Universidad de Granada, Granada 18071, Spain; e-mail: kawtar@ correo.ugr.es.

Manuel I. Capel, is in the Universidad de Granada, where he studied Physics, obtained the MSC degree in 1982 and the PhD in Computer Science in 1992. He worked in the Universidad de Murcia before moving in 1989 to the Universidad de Granada where he is now leading a research group in the field of Concurrent Systems. His e-mail address is: mcapel@ugr.es and his Web-page can be found at http://lsi.ugr.es/~mcapel (Ph: 0034958242816, Fax: 0034958243179).

A. *Basic concepts*

UML-RT adds four new building blocks to the standard UML meta-model. Three of them (capsules, ports and connectors) are used to model the structure of the system, and the fourth (protocols) models the communications within the system. The behavior of system components is modeled using Statecharts. These diagrams contain state variables and describe the changes to the states by syntactic expressions in some programming language. Capsules model complex software components that could be concurrent and physically distributed. The internal structure of the components is described by sub-capsules and the connections between these. A component interacts with its surroundings, and with its sub-capsules, through a set of ports which are the only parts of a component that are visible to other objects. The ports can be connected either to a Statecharts diagram defining the functionality of the component, or to the port of a sub-capsule. Therefore, a message sent to a port can be handled directly by the capsule, or forwarded to a suitable sub-component. Fig. 1 shows an example depicting a simple UML-RT component architecture that includes these concepts.

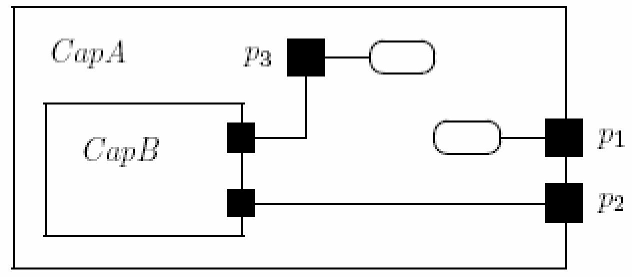


Fig. 1 An example of UML-RT concepts

The port p1 of the capsule CapA is connected to a Statecharts diagram, while p2 is connected to the sub-capsule CapB. In addition, the capsule has an internal port p3, i.e., the one which is only visible inside the capsule, connecting a port of the sub-capsule with a Statecharts diagram. A protocol defines a number of participating roles and the signals sent and received by each role. It can also contain a specification of the valid sequences of signals, which are encoded as in Statecharts. If no such specification is given, any sequence is considered valid. Connectors are used to model communication channels between two or more ports, the later ones must realize different roles of their mutual protocol. The protocols and connectors define the behavior of the system at the architectural level. UML-RT serves to model real-time systems and compared to standard UML, it provides some additional support when modeling the architecture of interactive systems, nevertheless it does not provide support for modeling timing issues. Only are timeouts allowed as time structures in a UML-RT model to introduce timing constraints in the system specification, but they are not supported by the modeling language itself. Consequently, UML-RT does not facilitate reasoning about the temporal properties of the system model. In reference [12] UML-RT is alternatively defined by giving a formal semantics of its entities, being the structural and behavioral parts of a UML-RT model semantically defined in terms of flow-graphs. This approach could be considered close to ours but it does not address concurrency issues as we do by using the programming notation CSP+T.

B. *UML profile for Schedulability, Performance and Time*

In 1999, OMG issued a request for proposals regarding a new UML profile addressing specific problems related to the development of real-time systems [13]. The main aim of the profile is to allow the exchange of models between different modeling tools, and between tools for modeling and analysis. It is also supposed to support essentially any type of analysis method, including many schedulability and performance analysis methods. These two requirements are somewhat contradictory, since schedulability and performance analysis methods typically consider only a particular domain, thus making assumptions about the underlying models of time, concurrency, etc. Thus, to include a new analysis method to the profile framework, the method provider must define the attributes that are essential of his method, and connects them to the appropriate models of resources and time. On the other part, when applying the method to a model, the developer may need to iteratively transform the latter one into an analyzable format containing all the information appropriate for the method. In order for the method to be useful, much of this transformation should be done automatically.

C. *Modeling Time*

The profile distinguishes between two types of time metric: physical and simulated. Physical time is considered to be continuous, dense, unbounded and fully ordered, while simulated time models the timing concepts as visible entities from a system viewpoint. Time can be discrete or dense, and possibly non-monotonic. Simulated time can be associated with physical time by means of periodic reference clocks, which group temporally close physical time instants and associates their occurrence to the same clock tick, according to some given time granularity. To allow simulated time to be used in models, the UML-RT profile contains definitions of timers and clocks. A timer generates a certain timeout event when a specified time instant is reached, while clocks periodically generate clock tick events. A clock or timer is always associated with a reference clock that provides the (simulated) time. They also have a number of attributes in common, such as resolution and drift.

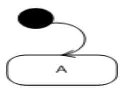
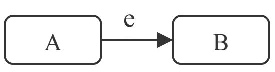
D. *Modeling Schedulability*

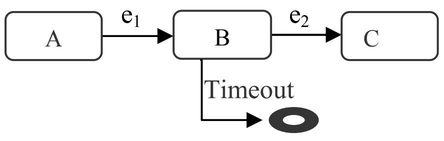
The UML-RT profile describes a set of common scheduling annotations, which are sufficient to perform basic schedulability of real-time tasks. It is expected that individual tool vendors provide specialized annotations to allow for more extensive analysis. The annotations defined by the profile include priority, absolute and relative deadlines, and worst-case completion time.

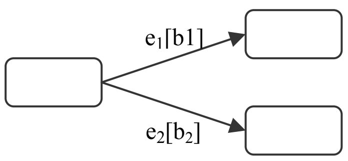
III. Formal Specification and Transformation Methodology from UML-RT

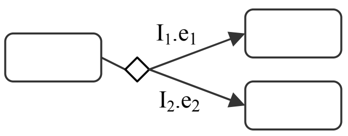
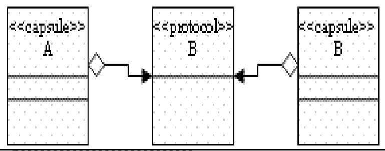
We are interested in systematically performing the specification of the behavioral and structural aspects of a real-time system. These two “different” specifications are usually attained in UML by respectively using class diagrams and Statecharts diagrams. Our aim is to give a precise semantics to the system interactions (process communications, event occurrence, etc.) and its main operations related to time. We can do it by constructing a model of the system interactive behaviour as well as modeling its timing constraints. CSP+T is a real time specification language, which adds expressive power to some of the sequential aspects of CSP and allows the description of complex timing constraints and temporal dependencies among real-time processes. CSP+T describes a set of deterministic processes with time constrained behaviour, which constitutes a formal specification language of use for the design of the majority of real-time systems.

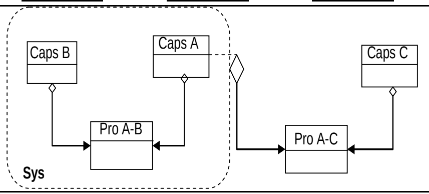
TABLE I
MAPPING RULES FROM UML-RT TO CSP+T

	StateChart Diagram + Class Diagram	**Description**	**CSP+T Model**
1.		Initial State	Sys = 0.∗ →A (∗: instantiation event)
2.		Transition from State A to State B triggered by a marker event e	A = e >< m_e→ B

3.		(e_1,e_2) two successive events, e_1 is a marker event and e_2 is its restricted event	A = e_1>< me_1 → B
			
			B = (I(e_1). e_2→ C | Timeout) → Skip.
			With the enabling interval I defined as:
			I (e_1) = [me_1, me_{1+}T], T∈ R^+.

4.1	External choice:
	
		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&b2→C)
			
			If (e_1≠ e_2) we can write :
			A= (e_1&b_1→B | e_2&b2→C)
			Operator □ represents non-deterministic and operator | represents deterministic choice.

4.2	Internal choice: 	The decision on which branch to take depends on the prior action within the same execution step	A= (I_1.e $_1$→ B) п (I_2.e_2→C) With the enabling intervals defined as: I_1 = [0, T1), T_1> 0 & I_2 = [T_1, T_1+T_2] T_2> 0.
5.		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	Sys = {A//B}\ {E_{AB}}
			The protocol common to capsules A and B is hidden from the environment
			
			Sys1={Sys//C}\{E_{AC}}

A series of transformation rules, already discussed in [6], will allow us to create a CSP+T model from the UML analysis model of a real-time system. These transformation rules can be considered the core of our complete top down systematic specification technique. Each object in a class diagram is defined by a process which behaviour is described by a Statecharts diagram. In general, we will follow a transformation process that consists of the following steps:

1. First of all, we define the dynamic behaviour of all components in the system using Statecharts, 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, do the following steps:
      i. Find the marker events and the restricted ones
      ii. Assign a special function `gettime()` 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 e2 does not occur within the enabling interval. See rule 3 of Table I as an example of this scenario

2. Transform each Statechart diagram into a CSP+T process.

a. Map each state into a CSP+T process, the initial state is assigned to a process term that includes the instantiation event (rule 1), which gives the global time origin

b. Transition from P to Q triggered by a marker event e is translated into the CSP+T process *P= e><te → Q*, being *te* 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 transitions. 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 chioce depends on the trigger event that occurs upon exiting from the current state (rule 4)

3. Create a class diagram for modeling the whole system to show the relation between system components:

a. Model all system components as capsules

b. Model capsules interaction 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 the internal events of the process

c. Protocol operation denotes the communication between two capsules, or in other case the signals 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 event (private operations) are hidden.

A. Example1

We give a simple example to provide the insight of these last steps (4.e-f):
Caps A interact with Cap B within a protocol Pro A-B, and with Caps C within a protocol ProA-C.

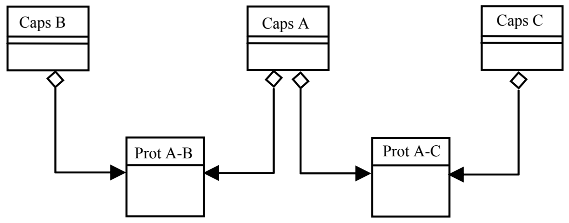


Fig. 2 Connection diagram of UML-RT capsules

Let Sys be the process that models the system composed by two capsules, CapsA and CapsB: `Sys= {CapsA // CapsB} \{Events`$_{\text{Prot A-B}}$`}`, and let Sys$_1$ be the process that models the system composed by Sys and CapsC: `Sys`$_1$`= {Sys // CapsC} \{Events`$_{\text{Prot A-C}}$`}`. Therefore, the method is compositional, i.e., the two subsystems represented as processes are encapsulated in the process term Sys1.

B. Timing Constraints

For two successive marker events (e1, e2), we assign to e1 (the preceding event) a marker variable v to record the time at which the event occurs and to e2 (the successor event) an enabling interval I [v, v+ T], with T being a time interval which takes enough time so that the event can occur (rules 3 and 4), meaning that the occurrence of e2 is restricted to the time T from the occurrence of event e1; or otherwise if the event does not occur within the enabling interval, a special event timeout is triggered to bring the system state to a null state (skip). In [6] can be seen the complete definition of marker events, variables and enabling intervals.

C. Modeling Class Diagrams

- *Modeling Protocol:* Each two capsules associated within a class diagram exchange a sequence of signals defined in protocol Pt (CN, Ep) , being CN a pair of the form (c1, c2) in which c1, c2∈ CS (set of capsules) and c1≠c2, and Ep is a set of events shared between two capsules.

- *Modeling Associations in Class Diagrams:* an association in a class diagram is modeled as a parallel composition in CSP+T made up of two capsules, having turned the events of its associated protocol into hidden events.

IV. THE PRODUCTION CELL CASE STUDY

The case study [14] presents a realistic industry-oriented problem, where safety requirements play a significant role and can be met by the application of formal methods. The manageable size of the Production Cell (PC) design task allows for experimenting with several approaches.

The PC processes metal blanks which are conveyed to a press by a feed belt. A robot arms takes each blank from the feed belt and places it on the press, then the robot arm withdraws from the press proximity, 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. 3).

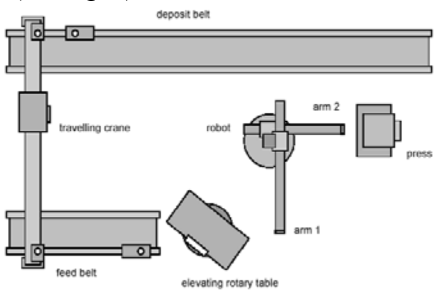


Fig. 3.Production Cell

This basic sequence is complicated by further details:

• To enhance the utilization of the press, the robot is fitted with two arms; thus, making it possible for the first arm to pick up a blank while the press is forging another plate.

• The robot arms are placed on different horizontal planes, and they are not vertically mobile. This explains why an elevating rotary table has to be put in between the feed belt and the robot.
• Another consequence of the fact that the two robot arms are at different levels is that the press has three states: (1) open and prepared to be unloaded by the lower arm, (2) open to be loaded with a metal plate by the upper arm, and (3) closed while pressing.

A. *Modeling the Robot*

The robot comprises two orthogonal arms. For technical reasons, the arms are set at two different levels. Each arm can retract or extend horizontally. Both arms rotate jointly. Mobility on the horizontal plane is necessary, since elevating rotary table, press, and deposit belt are all placed at different distances from the robot's turning center.

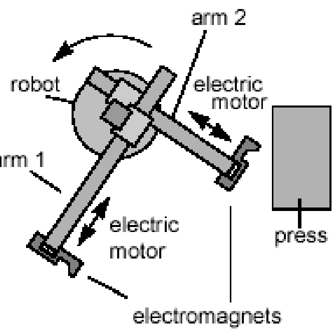


Fig. 4 Robot and press (top view)

The end of each robot arm is fitted with an electromagnet that allows the arm to pick up metal plates. The robot's arm task consists in taking metal blanks from the elevating rotary table to the press and transporting forged plates from the press to the 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, then Arm 1 extends and picks up a metal blank from the table.
2. The robot rotates counterclockwise until Arm 2 points towards the press, then Arm 2 extends and picks up a forged piece from the press.
3. The robot rotates counterclockwise until Arm 2 points towards the deposit belt, then Arm2 extends and drops the piece into the belt.
4. The robot rotates counterclockwise until Arm 1 points towards the press, then Arm 1 extends and places the blank on the press.

The Robot Class Diagram, Fig. 5, shows the robot architecture, the interaction between the robot controller and the two arms of the robot. Its shows the structure of the robot, its classes and their associations, but it does not describe the behavior of the class instances. We use UML Statecharts to model the behavior of the robot controller and the 2 robot arms.

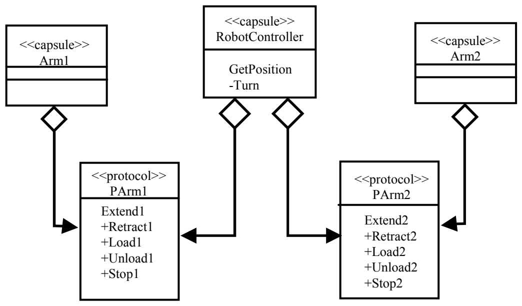


Fig. 5 The Robot component class

By applying some of the mapping rules in Table I to a Robot Statecharts diagram we obtain the interactive and temporal behaviour specification of the Robot-controller, specified as a CSP+T process term.

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):

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

By composing in parallel the processes *RobotController-Arm1* with *Arm2* we obtain the Robot process structure:

```
Robot =
(Robotcontroller-Arm1 // Arm2) \ {A2Extend,
A2Retract, A2Load, A2Unload, A21Stop}
```

To avoid collision between arms and other PC components (press, belts, etc.), some of which have to be to loaded or unloaded, we store in a variable *tposx* the time at which the robot arrived to an given position in each composite state of the robot. We assign an interval *I[tposx, tposx+TCL/U]* 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 occur within the enabling interval, or otherwise the timeout event is triggered and the robot exits the actual 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 Statechart diagram in Fig. 6 shows the integration of these concepts, needed to describe the PC safety requirements.

B. *Production Cell Diagram*

The Production Cell Class Diagram (Fig. 7) shows the association between the objects and the events shared between any two objects communicating in the PC protocol.

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 every 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}
```

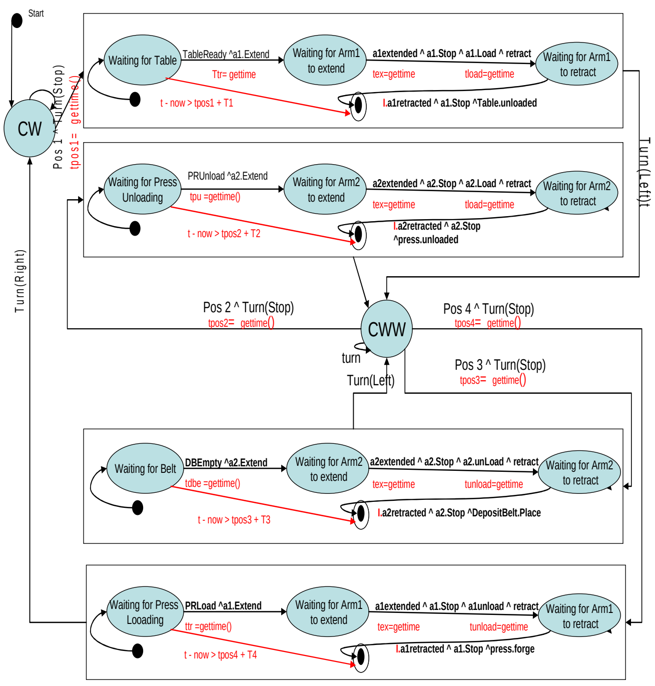

Start
CW
CWW
Waiting for Table
TableReady ^a1.Extend
Ttr= gettime
Waiting for Arm1 to extend
a1extended ^ a1.Stop ^ a1.Load ^ retract
tex=gettime
tload=gettime
Waiting for Arm1 to retract
t - now > tpos1 + T1
l.a1retracted ^ a1.Stop ^Table.unloaded
Pos 1 ^ Turn(Stop)
tpos1= gettime()
Waiting for Press Unloading
PRUnload ^a2.Extend
tpu =gettime()
Waiting for Arm2 to extend
a2extended ^ a2.Stop ^ a2.Load ^ retract
Waiting for Arm2 to retract
t - now > tpos2 + T2
l.a2retracted ^ a2.Stop ^press.unloaded
Turn(Left)
Turn(Right)
Pos 2 ^ Turn(Stop)
tpos2= gettime()
Pos 4 ^ Turn(Stop)
tpos4= gettime()
Pos 3 ^ Turn(Stop)
tpos3= gettime()
turn
Turn(Left)
Waiting for Belt
DBEmpty ^a2.Extend
tdbe =gettime()
a2extended ^ a2.Stop ^ a2.unLoad ^ retract
tunload=gettime
t - now > tpos3 + T3
l.a2retracted ^ a2.Stop ^DepositBelt.Place
Waiting for Press Looading
PRLoad ^a1.Extend
ttr =gettime()
a1extended ^ a1.Stop ^ a1unload ^ retract
t - now > tpos4 + T4
l.a1retracted ^ a1.Stop ^press.forge

Fig. 6 Robot Controller Statecharts diagram

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

The hidden events, in the above specification, correspond to the private operations appearing in the corresponding class diagram.

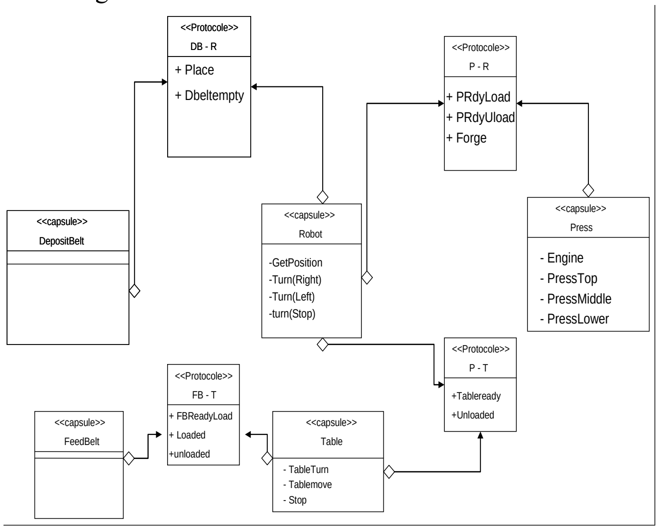


Fig. 7 Cell Production diagram

V. Conclusion

In this paper, we describe a systematic method to derive a 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 imprecision that UML models present in describing real-time systems. The future and ongoing work in our project is aimed to use the proposed method for automatic code generation of embedded control real-time systems and to attain integration and interoperability with state-of-the-art UML-RT software tools, such as the ObjectTime [10] one.

Acknowledgment

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

References

[1] B.Selic and J.Rumbaugh, "UML for modeling complex real-time systems". Technical report, ObjectTime, 1998.
[2] John J.Zic, "Timed constrained buffer specifications in CSP + T and timed CSP". ACM Transaction on Programming Languages and Systems, vol.16, 6, 1994, pp. 1661-1674.
[3] A.W.Roscoe. "The theory and practice of concurrency". Prentice Hall, 1997.
[4] C.A.R. Hoare, "Communicating Sequential Processes", Prentice- Hall, 1978.
[5] D. Harel and A. Naamad, "The statemate semantics of Statecharts". *ACM Transactions of Software Engineering and Methodology*, vol.5, 4, October 1996, pp.293-333.
[6] M. I. Capel, J. A. Holgado, "Transforming SA/RT Graphical Specifications into CSP+T Formalism - Obtaining a Formal Specification from Semi-Formal SA/RT Essential Models", ICEIS 2005, vol.3, Proceedings of the Seventh International Conference on Enterprise Information Systems, Miami, USA, May 25-28, pp.65-72.
[7] M. I. Capel, J. A. Holgado, A. Escámez, "An Integration Scheme for CPN and Process Algebra Applied to a Manufacturing Industry Case", Modelling, Simulation, Verification and Validation of Enterprise Information Systems, Proceedings of the 3rd International Workshop on Modelling, Simulation, Verification and Validation of Enterprise Information Systems, MSVVEIS 2005, Miami, FL, USA, INSTICC Press, 2005, pp. 39-48.
[8] M.Y.Ng and M. Butler, "Tool Support for Visualizing CSP in UML", in *Proceedings of International Conference on Formal Engineering Methods*(ICFEM), Shanghai, China, 2002, pp. 287-298.
[9] G. Booch, J. Rumbaugh, and I. Jacobson, "The Unified Modeling Language User Guide", Addison-Wesley, Reading, Massachusetts, USA, 1999.
[10] B. Selic, "Using UML for modeling complex real-time systems". Lecture Notes in Computer Science, 1474, Springer-Verlag, 1998, pp.250–260.
[11] B. Selic, G. Gullekson, J. McGee, and I. Engelberg, "ROOM: An object-oriented methodology for developing real-time systems", in *Proceedings 5th Int. Work. Computer-Aided Software Engineering*, July 1992, pp. 230–240.
[12] R. Grosu, M. Broy, B. Selic, and Gh. Stefanescu, "Towards a calculus for UML-RT specifications", in *Proceedings Seventh OOPSLA Workshop on Behavioral Semantics of OO Business and System Specifications*, Vancouver, Canada, October 1998.
[13] OMG, "Response to the OMG RFP for schedulability, performance, and time", June 2001. Available: OMG document number: ad/ 2001-06-14, http://www.omg.org/cgi-bin/doc?ad/2001-06-14.
[14] C. Lewerentz and T. Lindert, "Formal Development of reactive Systems: Case Study Production Cell". Lecture Notes in Computer Science, S 891, Springer-Verlag, Heidelberg, 1995.
[15] P.Welch, "Process Oriented Design for Java: Concurrency for All", in *Computational Science* - ICCS 2002, Lecture Notes in Computer Science, 2330, Springer-Verlag, April 2002 (Keynote Tutorial), pp. 687-687.