A SEMANTIC FORMALIZATION OF UML-RT MODELS WITH CSP+T PROCESSES APPLICABLE TO REAL-TIME SYSTEMS VERIFICATION

Manuel I. Capel Tuñón¹, Luis E. Mendoza Morales ², Kawtar Benghazi Akhlaki¹, and Juan A. Holgado Terriza¹

¹: Departamento de Lenguajes y Sistemas Informáticos
Escuela Técnica Superior de Ingeniería Informática y de Telecomunicaciones
Universidad de Granada
Periodista Daniel Saucedo Aranda s/n. 18071 Granada
e-mail: {benghazi, mcapel, jholgado}@ugr.es, web: http://lsi.ugr.es/~sc

²: Departamento de Procesos y Sistemas
Edificio de Matemáticas y Sistemas
Universidad Simón Bolívar
Apartado postal 89000, Caracas 1080-A, Venezuela
e-mail: lmendoza@usb.ve, web: http://www.lisi.usb.ve/

Key words: UML-RT, CSP+T, Formal specification, Real-time systems

Abstract. When a semi-formal language (like UML-RT) and a formal specification language (like CSP+T) are combined to obtain a Real-Time System (RTS) software specification, it is important to ensure the possibility of verifying the derived system design. A formal semantics of CSP+T temporal operators in terms of CSP trace semantics is presented here, and also a compositional refinement of CSP+T process terms to perform the system’s software verification. The semantics of the system requirements is preserved from their initial specification to the final system implementation. The inclusion relationship between the traces of increasingly refined process terms assures semantics preservation.

1 INTRODUCTION

The specification and design of a real-time system (RTS) is a complex task involving the description of the behaviour, intercommunication and architectural aspects of the system. In order to fulfill these requirements, UML and ROOM (Real-Time Object-Oriented Modeling language) are combined as UML for Real-Time (UML-RT) [1]. UML introduces four new constructors into UML-RT: capsule, protocol, port and connector, by using stereotype mechanisms [2, 3]. A UML-RT model of the system is a set of diagrams and system properties. In our approach, we choose diagrams that mainly represent the following architectural views: static data, dynamic behaviour, and dynamic object relationships, expressed, respectively, by class,
state and structure diagrams. Nevertheless, some fundamental constructs for modelling non-
functional requirements and temporal constraints of RTSS have been omitted from UML-RT.
Although UML syntax is well understood and widely accepted in the industry, it lacks a formal
semantics. On the other part, it is well known that the combination of UML modelling notation
with formal specification languages [4, 5] may benefit from the complementary features of both
notations, whilst their individual awkwardness as regards modelling some aspects of real-time
and concurrent systems can be mutually compensated. To obtain a provably correct formal
specification of the target system, we propose a transformation scheme, between the analysis
entities of UML-RT [1] and their representation as correct terms of the CSP+T, [6, 7] a formal
specification language, based on a set of rules.

Our transformation scheme is performed by means of a set of sound mapping rules already
established in previous works [8, 9, 10, 11]. This result is obtained in the second phase, by
applying a bottom-up strategy. To establish the formal basis of the methodological approach,
we seek to prove that the combination of the refinement concept of UML-RT and the process
composition concept of CSP and CSP+T guarantees that correctness is maintained during the
application of the method. Trace semantics [12, 13] of CSP and a new interpretation of CSP+T
operators provides a firm basis on which to build a verification system. One goal of the present
study is to show how our methodological approach, using compositional refinement, opens up
the possibility of verifying a real-time software system. The rest of this paper is structured as
follows: section 2 explains the CSP+T specification language features and formal semantics of
CSP and CSP+T that support the transformations rules proposed in previous works, and dis-
cusses our concept of process term refinement, providing the semantics basis for carrying out
a compositional verification of the target system; section 3 describes the system specification
method that we formalize here; section 4 presents the application of the method to the specifica-
tion of a key Production Cell component, known as Press. Finally, some conclusions are drawn
and a reference list for further information is provided.

2 CSP AND CSP+T

CSP (Communicating Sequential Processes) [12, 14] is an event-based notation primarily aimed
at describing the sequencing of events within process behaviour and the synchronisation (or
communication) between processes. CSP is based on a theoretical calculus that provides us
with a set of mathematically well-defined process terms (valid concurrent processes) derived
from an abstract grammar which includes operators to express the concurrent, non-deterministic
composition of these terms. Theoretical CSP and other similar process description formal lan-
guages have been called process algebras. If the execution of a single operator succeeds, it can
have an effect on the internal state of the component (assignment operator), or on the compo-
nent’s environment (output operator) or on both (input operator). The execution of a structured
operator implies: concurrent composition (∥), alternative (|), repetitive (*), guards (→), external
choice (□) (deterministic selection of one open guard within a structured alternative command),
internal choice (∩) (non-deterministic selection), hiding of communication actions sets (\) and
renaming ([A ↦ B]) of syntactical symbols sets.
In the group of CSP derivatives to describe time intervals, Timed CSP [12, 13] and CSP+T [6, 7] are worth to be mentioned here. Timed CSP captures timing constraints (TCs) by means of parallel composition of a set of processes, each of which describes a specific TC. These processes are viewed as a refinement of system behaviour that includes temporal information into system executions, thus facilitating the formal specification and verification of software systems that depend on time. However, the algebraic manipulation of complex timing requirements of a parallel composition of processes in a system may, in some cases, be impossible to perform without extending Timed CSP for allowing the representation of specific events occurring during a system execution. On the other hand, CSP+T [6, 7] permits us to describe TCs of processes by sequential syntactical terms without resorting to parallel composition of processes, thereby reducing the need for adding any parallel processes to express TCs. Furthermore, by properly using the additional language’s expressive power, we are allowed to render some complex parallel system into sequential processes.

2.1 CSP trace semantics

The execution behaviour of syntactical process terms in CSP can be represented by the set of all sequences of events, which are called traces. There are three semantics (traces, failures and divergences) of CSP process terms, which are discussed in detail in [12, 14]. For reasons of space, here we can give only a brief review of the fundamentals of trace semantics. The syntactical terms represent deterministic processes, since they are generally not affected by divergences of failures. For concatenation, filtering, renaming of the symbols appearing on sequences of events, we have the functions \( \land, \oplus, \mapsto \), respectively. Concatenating a pair of sequences \((a, b)\) constructs a new sequence by simply putting them together in this order. The result is denoted \( a \land b \). The length of the resulting sequence \( \#(a \land b) \) is equal to the sum of the length of \( a \) and \( b \). Within a nonempty sequence \( t \), its first symbol is denoted \( t_0 \) (head), and the result of removing the first symbol is denoted \( t' \) (tail). The filtering function \( \ominus \) is deployed to filter away elements of a given sequence. By \( A \ominus t \) we denote the sequence extracted from \( t \) by removing all its elements that are not in the set of elements \( A \). The change of symbols (renaming), denoted \( t[A \mapsto B] \) is a function which maps a sequence \( t \) of symbols in \( A \) into another sequence of symbols in \( B \) by applying the mapping \( (\mapsto) \) to each element of the sequence. In general, renaming does not distribute through filtering, i.e. \( (C \ominus t)[A \mapsto B] \neq C \ominus (t[A \mapsto B]) \), unless the mapping is a one-one function (injection). There is a calculus defined to calculate the \( traces(P) \) of any single process term \( P \) in CSP by applying the trace-semantic interpretation of the constant processes (SKIP, STOP) and the operators of the process algebra. The non-deterministic \( (\sqcap) \) and the general choice \( (\sqcup) \) operators of CSP cannot be distinguished by their traces. Nevertheless, these two operators present a different behaviour in general. It is possible to put them in an environment in which \( P \sqcap Q \) can deadlock at this first step, but \( P \sqcup Q \) cannot.
2.2 CSP+T Trace Semantics

CSP+T is a new real-time specification language which extends the CSP constructs with temporal primitives in order to enable the description of complex event timings from within a single sequential process. The calculus of CSP+T traces of any single process is carried out by adding a trace-semantic interpretation of the new operators which are not present in CSP. These operators are as follows:

1. **Process instantiation (\(\ast\)-star-).** A process must be instantiated before being executed in CSP+T. The specification of \(P\), e.g. \(P = a \rightarrow \text{STOP}\), becomes, \(P' = 1.\ast \rightarrow s.a \rightarrow \text{STOP}\) where \(s\) is a time stamp and \(s \in [1, \infty)\). The event \(a\) occurs only once in the interval (see trace semantic 1 in Table 1).

\[
\begin{align*}
1 & \quad \text{traces}(0, s \rightarrow a \rightarrow P) = \{t \mid (t = \emptyset) \lor (t_0 = \ast \land s(t_0) = 0 \land s(a) \geq 0 \land t' \in \text{traces}(P))\} \\
2 & \quad \text{traces}(0, s \rightarrow a \land v \rightarrow P) \quad \text{where} \quad t' = a \land v \rightarrow P. \\
3 & \quad \text{traces}(0, s \rightarrow a \land v \rightarrow I(T, v), c \rightarrow P) \quad \text{where} \quad t' = a \land v \rightarrow I(T, v), c \rightarrow P, \\
4 & \quad \text{traces}(0, s \rightarrow a \land v \rightarrow I(T, v), c \rightarrow P) \quad \text{where} \quad m = \min(T, T_v). \\
5 & \quad \text{traces}(0, s \rightarrow a \land v \rightarrow I(T, v), c \rightarrow P) \quad \text{where} \quad s_1 \| s_2, 0.\ast \rightarrow b \land v \rightarrow I(T, v), d \rightarrow Q. \\
6 & \quad \text{traces}(0, s \rightarrow a \land v \rightarrow I(T, v), c \rightarrow P) \quad \text{where} \quad \exists s \in [1, \infty), s \leq 1 \land s \neq 1. \\
\end{align*}
\]

Table 1: Trace semantics of CSP+T operators

2. **Marker variable (\(\infty\)).** \(ev \infty v\) means that the time at which \(ev\) is observed is recorded in the marker variable \(v\). Within a sequential process, successive events form a non-decreasing monotonic sequence. If we specify the process \(P = 1.\ast \rightarrow a \land v \rightarrow \text{STOP}\), for each process execution, the marker variable \(var\) will always satisfy \(var > 1\). The scope of marker variables is strictly limited to one sequential process (see trace semantic 2 in Table 1).
3. Event-enabling interval \( I \). 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. The specification of 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), is \( P = 0. \cdot \rightarrow [1, 2].a \sqsupset \triangleright v \rightarrow \text{STOP} \). 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 \( v \) and \( T \) define the duration of the time interval starting at the time instant stored in \( v \) (see trace semantic 3 in Table 1).

4. Parallel composition of two processes. In addition to the communication alphabet, the behaviour 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 (see trace semantic 4 in Table 1). In the case of the communication alphabets are not equal and the enabling intervals are partially overlapping, different options can occur (see trace semantic 5 in Table 1). Especially when does not exist matching communications between both processes –there are not common time instants at which the events in the intersection alphabet can occur–, the parallel composition acts as the STOP process (see trace semantic 5 in Table 1). When the communication alphabets are not equal and the enabling intervals are disjoint, then the interleaving takes place (see trace semantic 6 in Table 1).

3 FORMAL SPECIFICATION AND TRANSFORMATIONAL METHODOLOGICAL OF UML-RT TO CSP+T

The methodological approach is intended to help the designer with the complete specification of an RTSS. As can be seen in Figure 1, the proposed procedure is divided into two main phases: the first one (top-down modelling process) is intended to model the system using UML-RT, while the second one (bottom-up specification process) obtains the formal specification in terms of the CSP+T process algebra syntax.

The transformation starts from the UML-RT model and allows the integration of UML State Diagram (UML-SD) and collaboration diagrams in order to completely describe the behaviour of the system components (capsules) and to achieve the formal specification of real-time constraints defined on system processes. 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. The transformation is carried out by a system of rules [8, 9, 10], so that the final system design is correct by construction.
3.1 Refinement and compositional verification

According to UML rationale [3], we can say that refinement is a relationship that represents a fuller specification of something that has already been specified at a certain level of detail. In our case, we consider refinement as a means of adding information to UML-RT diagrams such that this new system specification becomes closer to an implementation of the intended system, which is usually affected by hard real-time constraints. A firm foundation for compositional verification can therefore be obtained [15].

The concept of refinement between process terms of CSP or CSP+T is analogous of that of satisfaction (|=). If a specification S logically implies another specification T, i.e. S ⇒ T, then for every process P:

\[ P \models S \Rightarrow P \models T. \]

If P is considered an implementation of a system which meets its specification S, i.e. P |= S, this means that every possible observation on a execution sequence of P must also appear on at least one of the sequences described by S, or more formally,

\[ \forall t \in \text{traces}(P) \Rightarrow S. \]

What gives the basis to define CSP syntactical terms refinement is that a process P’ is a refinement of another process P, i.e. P ~ P’, iff given the specification S—which is satisfied by P—, the following implication holds:

\[ \forall t \in \text{traces}(P) \exists t' \in \text{traces}(P') : t' \Rightarrow S. \]

The process term P’ is a more detailed description of the target system than the one represented by P, and P’ constitutes a specification closer to the intended system implementation.

By applying the transformation rules introduced in [8, 9, 10] we obtain a refinement of UML-RT analysis entities into CSP+T process terms, which can be understood as a refinement restricted
to externally visible behaviour, also known as black-box refinement [15]. We may define the function,

$$bb\_refine : UML\_Entity \times \mathbb{P}(Comm\_act) \rightarrow Traces(cspTerm)$$

By applying the proposed method, a detailed system specification of the target system can be obtained. The system specification is made up of a set of CSP+T processes including the specification of temporal constraints. The system specification derived satisfies the initial system specification and can be considered an implementation of its requirements. As Figure 2 shows, the development of any system is carried out by a compositional refinement procedure, starting from an initial UML-RT class diagram of the target system. In each transformation step, a certain kind of diagram is complemented by another type of diagram that adds additional information to the previous system specification. For instance, collaboration diagrams provide information about the internal communication structure of those system components bound to execute concurrently, such as capsules. In turn, a UML-SD gives information about the internal control state of capsules, which describes and also defines the communication protocols used to communicate capsules through ports.

![Figure 2: Correctness preservation during the Method application.](image)

Each level of diagram during the refinement of the UML-RT models can be transformed into CSP+T process terms according to the rules given in [8, 9, 10]. Lower level processes refine these of the upper levels in the system design, i.e., let us consider $P_{1/2...n} = \{[UML - SD(i)]\}$, $P_1 = \{[Collaboration\,Diagrams]\}$, $P_0 = \{[Class\,Diagrams]\}$, where $P_0$, $P_1$, $P_i$ are structured process terms of CSP+T representing different views of UML-RT diagrams of a system, which unambiguously represent the different descriptions of the system behaviour during its specification and design stages. In order to prove that a detailed design obtained from the initial UML-RT model of the system is also correct, it must be shown that the following refinement relation between processes, $P_0 \leadsto P_1 \leadsto ||_{1/2...n}P_i$ holds. Furthermore, our scheme allows us a compositional verification of the target system. The compositional verification of the target system is a result of the fact that the satisfaction relation is closed w.r.t. the conjunction operator,

$$If\ P_i \models S_i \Rightarrow System \equiv (||_{i=1...n}P_i) \models (\bigwedge_{i=1...n}S_i)$$
so allowing us to carry out the compositional analysis of complex systems by compositional refinement.

4 THE PRODUCTION CELL CASE STUDY

The Production Cell is a standard example for evaluating methodologies for designing embedded systems [16]. It is a model of an actual industrial installation in a metal-processing plant in Karlruhe, Germany. 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 (see Figure 3). The task of the press is to forge metal blanks. The press consists of two horizontal

```plaintext
Interaction of the Press

PCPL = {Press_Top, Press_Bottom, Press_Middle, Moves_Top, Moves_Middle, Moves_Down, Moves_Stop}

Press internal structure

R1 = {Ppl \rightarrow Cpcpl}
R2 = {Ppc \rightarrow Cpcpl, Pp \rightarrow Crp}
Press = (PC[R1] || Plant[R2]) \ Cpcpl

Plant behaviour + timing requirements

Plant = Start \rightarrow Stopped
Stopped = (Moves_top \bowtie top \rightarrow Going_Top | Moves_Middle \bowtie tmd \rightarrow Going_Middle)
Going_Top = I(Top, top) \rightarrow Press_Top \rightarrow AtTop
Going_Middle = I(Tmiddle, tmd) \rightarrow Press_Middle \rightarrow Atmiddle
Atmiddle = Moves_Stop \rightarrow Stopped
AtTop = Moves_Down \bowtie tb \rightarrow Going_Bottom
Going_Bottom = I(Th, th) \rightarrow Press_Bottom \rightarrow AtBottom
AtBottom = Moves_Stop \rightarrow Stopped

Press Controller behaviour + timing requirements

PC = Start \rightarrow Waiting_L
Waiting_L = forge \rightarrow Moves_Top \rightarrow Pressing
Pressing = Press_Top \rightarrow Moves_Middle \rightarrow Unloading
Unloading = Press_Bottom \rightarrow Moves_Stop \rightarrow Waiting_U
Waiting_U = Unloaded \rightarrow Moves_Middle \rightarrow Loading
Loading = Press_Middle \rightarrow PrRunload \rightarrow Waiting_L

Figure 3: Specification of the Press
```
plates, with the lower plate being movable along a vertical axis. The press operates by pressing
the lower plate against the upper plate. Because the robot arms are placed on different horizontal
planes, the press has three positions. In the lower position, the press is unloaded by arm 2,
while in the middle position it is loaded by arm 1. The operation of the press is coordinated with
the robot arms as follows: first, the press opens in its lower position and waits until arm 2 has
retrieved the metal plate and left the press; next, the lower plate moves to the middle position
and waits until arm 1 has loaded and left the press; finally, the press closes, i.e. the metal plate
is forged. This processing sequence is carried out cyclically. The specification of the press is
obtained by applying our methodological approach. Figure 3, summarizes the application of
the modelling and specification procedure.

From the transformation of UML-RT entities into CSP+T, we obtain the CSP+T syntactical
terms by applying the mapping rules given in [9]: \( P_1 = \{ [\text{Press Collaboration Diagram}] \} = \text{Press}; \)
\( P_2 = \{ [\text{Plant UML – SD}] \} = \text{Plant}; \)
\( P_3 = \{ [\text{PC UML – SD}] \} = \text{PC}. \) Let \( S \) be the
Press requirements specification, so that \( \text{Press} \models S. \)

We have : \( \text{Press} = (\text{PC}[R1] || \text{Plant}[R2]) \setminus \text{Cpcpl}, \)
then \( \forall t \in \text{traces}((\text{PC}[R1] || \text{Plant}[R2]) \setminus \text{Cpcpl}) \Rightarrow \text{Press}, \) \( (1) \)
and also we have : \( \forall t \in \text{traces}((\text{PC} || \text{Plant})) \Rightarrow (\text{PC}[R1] || \text{Plant}[R2]) \setminus \text{Cpcpl}. \) \( (2) \)
From (1) and (2) we obtain : \( (\text{PC} || \text{Plant}) \Rightarrow (\text{PC}[R1] || \text{Plant}[R2]) \setminus \text{Cpcpl} \Rightarrow \text{Press}, \)
then \( \text{Press} \sim (\text{PC}[R1] || \text{Plant}[R2]) \setminus \text{Cpcpl} \sim \text{PC} || \text{Plant}. \)

These implications demonstrate that the parallel composition of the two processes PC and Plant
is a refinement of the CSP+T process named Press.

5 CONCLUSIONS

This paper presents the trace semantics of the CSP and CSP+T operators, and shows how our
methodological approach, by compositional refinement, opens up the possibility of verifying
a software system design. It is also shown that the combination of the refinement concept of
UML-RT and the process composition concept of CSP and CSP+T guarantees that correctness
is maintained during the application of the method. Finally, the approach is applied to the
Production Cell case study to show that the refinement relation between different abstraction
levels of the system specification holds and to describe how we can profit from this relation to
validate the intended system by compositional verification.

REFERENCES


Addison-Wesley, Reading, Massachusetts, USA, 1999.


