ATransformational Approach to the Systematic Design

of Real-Time Systems

Manuel I. Capel Tuñón, José R. Balsas Almagro

1Dpto. Lenguajes y Sistemas Informáticos. Universidad de Granada
2Dpto. Informática. Universidad de Jaén.
mcapel@ugr.es, jrbalsas@ujaen.es

Abstract. In this paper, a bottom-up formal technique to obtain a correct system specification from the RT/SA requirements specification of a real-time system is proposed. The systematic procedure yields a complete specification in terms of CSP+T processes by means of the application of a set of transformation rules, which give a formal semantics to the semi-formal analysis entities of RT/SA. The approach takes advantage of the long tradition of RT/SA methodologies in the industry and at the same time, aims to foster the use of Process Algebras as an adequate way to overcome the intrinsic imprecision that SA presents in describing real-time systems. The method has been developed so that it may be integrated into a Software Engineering Environment, such as a CASE tool or SA-based tool. Furthermore, a formal semantics is given to the RT/SA entities in terms of CSP+T process algebra, which, by means of the new CSP-based programming libraries in Java, provides our method with the capability of automatic code generation for several computing platforms. A detailed design of a complete solution to the production cell problem has been discussed so as to show how the method can be applied in order to solve a realistic industrial problem.

1 Introduction

Nowadays, the application of formal methods to Automated Software Engineering (ASE) environments is of paramount importance, since improving the technology in this field is considered fundamental for performing timing and dependability analysis of real-time systems without resorting to the construction of any expensive prototype of the final system. In this respect, we present a complete bottom-up systematic specification technique in order to derive a correct system specification from a semi-formal system requirements specification in RT/SA [2],[12],[5] by applying a set of transformation rules. It integrates two complementary approaches to describe a real-time system: (1) RT/SA based notations, and (2) CSP+T [14] process terms to model real-time processes including the specification of their timing requirements.

There are many proposals to obtain a formal system specification of real-time systems with a framework based on Process Algebras. Some of the most significant work on this subject was carried out by E.R. Olderog [9], who proposes a complete
Manuel I. Capel Tuñón, José R. Balsas Almagro

top-down systematic technique to derive a correct system design from a formal specification of system requirements built up from a set of trace logic formulas. One significant result of this work is the UniForM Workbench that provides tools for the development of reactive systems and allows the integration of formal methods and tools in a common development environment. We can find many tools in the industry nowadays, such as Simulink/Stateflow1, which claims to have a formal basis (Statecharts) and code generation capabilities, which can only partially address the features of a genuine formal tool capable of generating provably correct code. In Simulink/StateFlow, the verification of the final system is carried out by simulation.

RT/SA specification notations have been intensively used in the industry in recent decades. The practical use of these methodologies, however, leads to imprecisions and ambiguities, since SA notations lack formal semantics. Many proposals tried to overcome these inadequacies by complementing SA with a formal semantics. Among these it is worth mentioning the formalization of SA through Z and Larch [10], the translation from SA to Communicating Processes [4], the set of rules to give SA an interpretation by using H-L Petri nets [3]. However, as Baresi and Pezzè stated in their paper [1], all these proposals irremediably hurt the flexibility of SA by fixing a particular interpretation to SA entities, some of which should only have a weak semantics [5], [12].

Our approach therefore follows the guideline established in [1], and so the aim is to overcome the intrinsic imprecision that SA presents when describing real-time systems through the use of the CSP+T notation to describe unambiguously the different families of SA notations. However, the method does not fix a particular semantics when there are several possibilities to solve a given ambiguity and it will be up to the analyst to select the most convenient notation semantics depending on the system to be specified. This feature of the method can be attained because CSP+T provides primitives and composition operators which are capable of defining deterministic and non-deterministic notation structures. The proposed systematic derivation technique and the transformational rules can be easily integrated into state-of-the-art RT/SA software tools, such as AxiomSys [11], in order to provide a complete specification of an embedded or real-time control system, which in turn can be automatically translated to Java with the support of CTJ [6] or JCSP libraries [13]. The complete derivation process can be fully implemented through a library of pre-compiled patterns by following the same bottom-up strategy as in RT/SA when performing a system requirements specification.

The remainder of the paper is structured as follows. We first give some background on structured real-time system analysis methods, which is necessary to understand the transformation rules on which the derivation technique is based. In section 3, we completely describe the system specification method. In section 4, using the example of the Production Cell Control Specification, we present a complete system specification and how it can be carried out. Finally, the conclusions and the ongoing lines of work are presented.

---

1 The MathWorks, Inc.
2 Real-Time Structured Analysis

The methodologies and notations referred to under the generic denomination of Structured Analysis (SA) are mainly directed towards specifying the system behavior as a set of data transformations or processes, which describe the basic functions of the target system. There have been two parallel extensions to SA in order to address the specification characteristics of real-systems, both of which are completely equivalent. The first was proposed by Ward and Mellor (WM) [12], and the second by Hatley and Pirbhai (HP)[5]. A third variation, called the Extended System Modeling Language (EMSL), is an attempt to combine these two approaches to the structured analysis of real-time systems.

2.1 Production Cell Control Example

This well-known case study [8] presents a realistic industry-oriented problem, where safety requirements play a significant role and can be met by the application of formal methods. In the fundamental configuration, the production cell processes metal blanks that are conveyed to a press by a feed belt. The first robot arm takes each blank from the feed belt and places it in the press. Because the belt and the robot are different heights, there is an elevating rotating table which is designed to give blanks to the robot. The press forges a new metal blank and opens again. Forged metal plates are taken out of the press and put on a deposit belt by a second robot arm. Since the robot is fitted with two arms, the utilization of the press is enhanced, thus making it possible for the first arm to pick up the next blank while the press is forging another plate with the previous blank. A possible configuration of the production cell uses a traveling crane to return plates to the feed belt, so that the production sequence of the system can continuously function without an operator.

2.2 System Requirements Model (SMR)

The model consists of a hierarchy of transformation schemes rooted on the System Context Diagram (SCD). Each scheme “explodes” into a State Transition Diagram (STD) or into a Data Flow Diagram (DFD). The scheme denoted as SCD defines the border between the system, which should be understood as a double model describing
the data flow and the control flow relationships in the “solution domain”, and the environment, comprising the external entities (or terminators) to the system and representing the “problem domain”.

The RT/SA notations include other elements of representation, called analysis entities, Data Transformation Processes (DTPs), Control Transformation Processes (CTPs), Data Stores (DS), Control Stores (CS), Data Flows and Control Flows. Control Flows represent transportation of transient signals or events towards CTPs.

A transformation scheme in the RT/SA notations is represented by one SCD or one DFD. DFDs are composed of several copies of the above analysis entities and must include at least one DTP. DTPs change the input flows into output flows with no relation between the number of inputs and outputs. The same output can be sent to several analysis entities. A DTP must have at least one output flow. The bubbles that represent DFDs may explode into new, more detailed DFDs.

CTPs serve to transform input into output event flows. They cannot accept or generate any type of data flow, since a fundamental strategy of SA/RT is to separate the control and the data process descriptions within the system. A CTP is formally specified by means of a state transition diagram (STD). STDs should be deterministic Moore or Mealy automata, and they describe a sequence of state transitions of the system that cause the execution of DTPs to be triggered. Each transition comprises a condition (represented by an input control flow in the DFD) and an action that includes the activities to be carried out before the system reaches a new state. The internal action (τ) is not observable, and causes and apparently spontaneous change in the state of control of the system.

Data stores (DS) loosely represent data of a certain type that cannot be considered structured, and neither is it legal to suppose any type of extraction order of the data items from a data store. Destructive readings over a DS cannot be supposed as the data always remain in the store after readings. The DS cannot directly appear connected between them in diagrams, since the movement of data between DS must only be performed by the DTP. Control Stores (CS) can only store events of the same type, which are queued in FIFO order. Unlike the DS semantics, reading an event of a CS is a destructive operation. The DFD for the complete Production Cell Control System can be seen in Figure 3.
2.3 Flaws of RT/SA as a Specification Notation for Real-time Systems

The following ambiguities appear in both the WM and HP notations causing imprecisions in the specification, which can therefore draw non-predictability to the final system at a later development stage:

a) **Lack of any rule for defining primitive process specifications (PSPECs).** The only indication given is that these specifications should define the functional transformation performed by primitive DPTs. However, in realistic applications, DTPs do not only describe a purely functional behavior of processes, but, on the contrary, they often also include control and timing information.

b) **The enabling conditions of processes are not fixed.** The SA rationale is that processes are enabled whenever “sufficient data” appear in any of their input flows. However, the enabling conditions rules do not clearly indicate the expected behavior of a process when more than one of its input flows is carrying values.

c) **Execution time requirements for processes are excluded.** These requirements, when applied to practical cases, are used to specify either a maximum or minimum time to be associated with the execution of a process.

d) **The number and the type of the input flows entering a process are vaguely described.** When there are multiple input flows entering a process, it is necessary to define whether all the inputs must carry a value simultaneously to enable the process (synchronous case) or only a subset of the input flows (asynchronous).

e) **Simultaneous events awakening more than one transition.** This possibility is excluded in RT/SA notations since transitions exiting the same state are associated with different events, since STDs are Mealy machines. However, there should be no objection in allowing nondeterministic selection of transitions in notations for soft real-time systems.

All these imprecisions can be solved by giving a semantic interpretation to the SA entities that exclude any of these ambiguities. These interpretations can be easily programmed in CSP+T by using a set of rules, which translate each SA entity or scheme into a pattern that defines a CSP+ process.
3 Real-time Systems Specification with CSP+T

Many proposals have tried to overcome the problem of SA imprecision by complementing it with formal methods. The use of extensions of algebraic process description languages, such as CSP[7], CSP+T[14], the standard specification language LOTOS, etc. can give a precise and flexible interpretation to SA entities.

3.1. CSP+T

In the group of CSP derivatives to describe time intervals, we could mention Timed CSP [7] and CSP+T, the second one being a simpler approach than the first one. Although providing smaller descriptive power, but powerful enough to formally describe a set of deterministic processes with time constrained behavior, CSP+T is an adequate formal specification language for the majority of real-time systems.

The syntax of CSP+T is a superset of the syntax of the CSP. The main changes performed in CSP+T w.r.t. the original CSP are described as follows:

− Every process \( P \) defines its own set of communication symbols, i.e. its communication alphabet \( \alpha(P) \). These communications represent the events that the process \( P \) receives from its environment or internally occur (e.g. the null action \( \tau \)). External events can be understood to be a pure synchronization between one asynchronous process and its environment. Internal events, such as \( \tau \), are not externally visible. Any type of event causes a change of state of the process.

− The communication interface \( \text{comm}_\text{act}(P) \) of a given process \( P \) contains all the CSP-like communications \( (?, !) \) in which \( P \) can engage and the alphabet \( \alpha(P) \).

− A new operator, \( \star \) (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 the processes can start their execution. As an example, let us consider a process that initially can only engage in the event \( a \). This process in CSP would have been denoted as: \( P = a \rightarrow \text{STOP} \). In CSP+T, this process must be instantiated before executing itself. Given \( P' \), the timed version of \( P \), which is instantiated at time 1, \( s \) is a time stamp associated to \( a \), the specification of \( P' \) becomes

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

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

− A new event operator \( \triangleright < \) is introduced that is used jointly with a variable to record the time instant at which the event occurs, so that writing \( \text{ev} \triangleright < v \) means that the time at which \( \text{ev} \) is observed in a process execution is recorded in the variable \( v \). This time 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 had specified the process

\[
P = 1. \star \rightarrow \text{a} \triangleright < \text{var} \rightarrow \text{STOP},
\]

for each process execution, \( \text{var} \) will record the corresponding value of time at which the event \( a \) occurred, and it will always satisfy that \( \text{var} \geq 1 \). The variables associated to the operator \( \triangleright < \) are called marker variables and their scope is strictly limited to
A Transformational Approach to the Systematic Design of Real-Time Systems

One sequential process. They cannot be referenced or accessed in any other way in a concurrent composition of processes.

Each event is associated with a time interval, which is called the event-enabling interval. This interval represents the period of time over which the event is considered available to the process and its environment, and is relative to some preceding event of the current process execution. A process is considered to be the STOP process if it cannot engage in some alternative event after the expiration of the enabling interval of the event. The event-enabling intervals are continuous. 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, recording itself in the marker variable $v$ the time at which this event has occurred. The specification of this process is

$$P = 0 \cdot \ast \rightarrow [1,2] a \bowtie v \rightarrow \text{STOP}.$$ 

Therefore, after the process execution, the value of the marker variable will satisfy the inequality $1 \leq v \leq 2$. The enabling intervals are defined in terms of functions over a set of marker variables. When there are no marker variables referenced, the enabling interval is defined relative to the immediately preceding event.

$$P = \ldots E.P' \quad E = \{s \mid s = \text{rel}(x,v)\}$$

If the preceding event occurs at time $t_0$, then $\text{rel}(x,v) = [x + v - t_0, v - t_0]$, since the times for events are absolute and the times for intervals are relative to the preceding event. The semantics of the parallel composition of two processes with enabling intervals 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 in plain CSP. In the case of disjoint intervals, the parallel composition of processes behaves as the process STOP.

Finally, it should be noted that only deterministic processes can be described in CSP+T formal description language.

3.2 Generation of a System Specification from the SRM

In the sequel we will carry out the specification of real time systems using the CSP+T formal description language, starting from the SRM derived with the SA/RT method.

In order to obtain a CSP+T model of the system, it is necessary to represent every analysis entity of the SRM by a class of CSP+T processes. According to this approach, we intend to write a process CSP+T prototype for every DTP, CTP, DS, CS, continuous data flow, etc. A series of transformation rules will allow us to create a CSP+T model for every transformation scheme that appears in any diagram of the SRM. These transformation rules are the basis of our complete top-down systematic specification technique.

The interaction that a DTP or a CTP can provoke on their environments by discrete flows will be modeled as occurrences of events by means of TCSP-like communications, but the continuous flows need to be modeled by a specific rule. In general, we will follow a process that consists of the following steps:
1) Prepare the analysis schemes for carrying out the transformation. It may be necessary to rename some analysis entities to avoid conflicts (i.e., unwanted synchronizations) when constructing their model in CSP+T.

2) Transform the control transformation schemes (CTP) and data transformation schemes (DTP) of the lower level, i.e. those that do not explode into other schemes, into CSP+T processes.

3) Select the other schemes in ascending order, i.e. a CSP+T process for each Data Storage (DS), Control Storage (CS), Continuous Flow of Data, DTP or CTP that appear in the scheme, and build a CSP+T process for each entity within the scheme.

4) Once the CSP+T model has been obtained for all the entities in a scheme, one CSP+T process will be defined to model the complete scheme. If this scheme is already included in a CTP or a DTP of a higher level, repeat step (3), thus progressively integrating the CSP+T model of the system in an ascending way.

The process of hierarchic integration of transformation schemes finishes when the model of the System Context Diagram is obtained.

Definition 1. Given the set of RT/SA analysis entities $E$, proc an injective application, such that $P = \text{proc}(E) \in \text{CSP+T}$, we define

$$\text{Interface}(P) \subseteq \text{comm}_\text{act}(P) - \{\tau\},$$

as a set of actions that model the data or control flows on which the analysis entity interacts with its environment. $P$ is a syntactically correct process term of CSP+T that models the entity $E$.

Modeling process interface (rule 1). Interface($P$) is made up of an input communication symbol for every entity $O$, which is the origin of a communication towards $P$, and, vice-versa, of an output communication for every destination entity $D$. Where $O$ and $D$ are analysis entities with the only limitation being that both of them cannot be of type DS.

It is obviously necessary to rename when several entities $D_1, D_2, ..., D_n$ on a DFD accept the same input flow and, vice-versa, when several entities, $O_1, O_2, ..., O_n$ accept the same output flow. In RT/SA, it is irrelevant whether a set of entities synchronize or not at the time they accept a flow, but this is not the case in TCSP, since unwanted synchronizations could deadlock. The interface of control transformation process (CTP) is modeled in the same way by including events with a special meaning in comm_act. These are called $e$, $d$, $t$, after the RT/SA synchronization events enable, disable, trigger, that a CTP uses to control its associated DTPs.

Table 1. Transformation rules for RT/SA entities

<table>
<thead>
<tr>
<th>rule</th>
<th>RT/SA entities</th>
<th>CSP+T model</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1, 1.2.</td>
<td>Discrete data flow $f$ of $x$, with origin $O$ and target $T$. Or discrete event flow $e$.</td>
<td>$P = \text{proc}(f)$, $P = f ? x \rightarrow S$; $S = (f ? x \rightarrow S \mid f ! x \rightarrow S)$</td>
</tr>
<tr>
<td>2.</td>
<td>Continuous flow $f$ of $x$, with origin $O$ and target $T$.</td>
<td>$P = \text{proc}(f)$, $P = f ? x \rightarrow S$; $S = (f ? x \rightarrow S \mid f ! x \rightarrow S)$</td>
</tr>
<tr>
<td>3.1,</td>
<td>SDT defined as $(Q, C, A, T, q)$, $\forall (q_i, c, a, q_j) \in T$, $c \in C \cup {\lambda}$, $a \in A \cup {\lambda}$</td>
<td>$P = \text{proc}(q)$, $Q = \text{proc}(q)$</td>
</tr>
<tr>
<td></td>
<td>$a$ and/or $c$ are marker</td>
<td>$P = c \rightarrow (a \rightarrow Q)$ or $P = a \rightarrow Q$, $c = \lambda$ or $P = c \rightarrow Q$, $a = \lambda$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$P = a \rightarrow m_x \rightarrow (a \rightarrow m_x \rightarrow Q)$ or $P = a \rightarrow m_x \rightarrow Q$, $c = \lambda$</td>
</tr>
</tbody>
</table>
3.2. events with marker variables \( m_a, m_c \), or \( P = c?\langle m_c \rangle \rightarrow Q, a = \lambda \).

4. Data storage DS with input flows \( \{f_{i1}, \ldots, f_{im}\} \) and output flows \( \{f_{o1}, \ldots, f_{om}\} \)
   \[ P = \text{proc(DS)} \]
   Interface \((P) = \{f_{i1}, \ldots, f_{im}\} \)

5. Data Transformation Process DTP
   \[ P = \text{proc(DTP)} \] (a DTP can explode in additional entities).
   Interface \((P) = \text{interface (DTP)} \)

6. Control Transformation Process CTP
   \[ Q = \text{proc (CTP)} \lor Q = \text{proc (STD)} \] (Q can model the STD associated to a primitive CTP or its explosion)
   Interface \((CTP) = Q(\text{alphabet (Q)}-)\text{interface (CTP)} \)

7. \( E_1, E_2, \ldots, E_p \), RT/ST entities in the same schema S
   Interface(S) = \( E_i(\text{alphabet}(E_i)-\text{interface}(S)) || \ldots \)
   \[ E_p(\text{alphabet}(E_p)-\text{interface}(S)) \]

**Modeling continuous data flows (rule 2).** Continuous data flows cannot be directly modeled by means of communication events in CSP+T, since in CSP+T the communication is understood to be a synchronous message passing between 2 processes and a continuous flow of data denotes an uninterrupted communication between different processes. It will therefore be necessary to write an extra process (named \( S \) in the rule) for each continuous data flow that continuously offers the possibility of engaging in that communication to the other processes.

**Modeling State Transition Diagrams (rule 3).** Every CTP, called \( P \), of the lower level in the SRM hierarchy is represented by a unique STD from the point of view of control specification. An STD can be defined as a tuple \((Q, C, A, T, q)\) in which,
- \( Q \) is a set of states;
- \( C \) is a set of conditions, i.e., every condition denotes the occurrence of an external event, which corresponds to an input flow of control in \( P \), or to the occurrence of an internal event which is different from any internal control flow in \( P \), such as the internal action \( \tau \);
- \( A \) is a set of actions. An action causes the execution of an activity in the system. It can be easily identified since it corresponds to an output control flow in a DTP, or the occurrence of an internal event of a STD;
- \( T \) is a set of transitions. A transition is a tuple of the form \((q_1, c, a, q_2)\) in which \( q_1, q_2, c \in Q \), \( a \in A \) or is null, \( c \in C \) or is null, and its interpretation is: if in the state \( q_1 \), the condition \( c \) is satisfied, the action \( a \) will be performed and also a change to the state \( q_2 \) will occur. Notice that either \( c \) or \( a \) can be null;
- \( q \) is designated as the initial state of the STD, and \( q \in Q \).

The transition concept can be extended to specify timing constraints in the system by describing enabling intervals and marker events.

**Timing constraints.** These constraints will be described as a set of \( R \) of tuples \((e_i, I, e_j)\) in which \( e_i \in C \) or \( e_j \in A \), and \( e_i \) receives the name of the marker event, \( I \) is a real number interval of the form \([\alpha, \beta]\), where \( \alpha, \beta \in \mathbb{R}^+ \), and \( \alpha \leq \beta \) or \( I \) is an interval

\[2\] it is not a recursive definition, \( \text{interface (CTP)} \) on the right side of the equation is previously calculated by rules 1
relative to the preceding event or to the event $e_1$, $I(e_1)$ will denote the interval $I$ in the following text and $e_2 \in C$ or $e_2 \in A$ receives the name of a restricted event. The interpretation of a timing constraint $R$ is as follows: event $e_2$ can only occur within the interval of time $I$ from the occurrence of the event $e_1$, where both events can represent the satisfaction of a condition $c$ or the execution of an action $a$.

If the restricted event coincides with the condition $c$, this means that the condition will be satisfied during the interval of time $I$ to which it is restricted, the satisfaction of the condition outside the interval is not considered. In the case of the restricted event being action $a$, the system will be forced to carry out this action within the interval $I$ to which it is restricted, or otherwise the process in which the restricted event is programmed fails.

**Modeling timed Control Transformation Processes.** We shall therefore consider a timed STD as the tuple $(Q, C, A, T, q, R)$, i.e. an initial STD plus the timing constraints imposed on the system. The process that models the STD will be the process associated to the initial state of the system, i.e. it is activated when the system starts.

**Modeling data and control storages (rule 4).** A Data Store (DS) in the SRM is simply a class of entities capable of storing pieces of information for which we cannot assume any structure or formal definition. Therefore, no mechanism to specify data or to retrieve/insert data from/into a DS has been anticipated in the SRM of a system.

In our system specification model, a DS or a CS will be modeled as a CSP+T process capable of accepting information by communicating with other processes, or capable of offering its stored data through another communication. In order to represent DS entities by CSP+T processes, we need to assume some structure on the data stored in a DS. A DS modeled as a CSP+T process will therefore be able to hold information about every algebraically specifiable data type. In addition, the specification of DS can contain timing restrictions.

**Modeling of Transformation Specifications (rules 5, 6).** There is no agreement on how to perform the correct specification of a PSPEC (primitive DTP) in RT/SA. The specification of PSPECs is usually carried out in pseudocode, structured English, pre/post-conditions, etc. In this respect, we will suppose that the functionality of primitive DTPs is simple enough to allow us to obtain a model for each DTP as a single CSP+T process.

**Hierarchic integration of the entities of a diagram (rule 7).** Since we follow a bottom-up design method, we will begin by applying the above rules to the lower level schemes of the SRM of the system. When all the entities in which a diagram explodes have been modeled, we will be able to obtain the complete diagram, represented by a complex CSP+T process term, from its component processes. The method functioning is based on an iterative composition of the constituent processes and on the abstraction of their internal communications. The iterative process finishes when the context diagram of the system is obtained.
4 Complete System Specification Example

Let us first present a detailed modeling of the Robot Control Process (RC) of the production cell control system, since it is the process with the richest functionality among the processes that make up the design of the production cell. We will then model the table, press, and belts control processes. Finally, the integration of all the derived schemes will be obtained to show that the interface of the unique process coincides with the data+control flows shown in the SCD of figure 2.

Robot Turn Control
The objective of this process is to ensure that the robot arms can reach the positions established as safety positions in the production cell specification [8], i.e., no collisions between robot arms and belts, blanks cannot fall from the table or the press, etc. The robot’s safe positions are named in the sequel, robot_pos_0 (initial position with arms retracted), robot_pos_1 (arm 1 is placed in front of the rotating table and is prepared to extend and pick up a blank), robot_pos_2 (arm 2 points towards the press and is prepared to initiate picking up a plate), robot_pos_3 (arm 2 points to the deposit belt to place a plate on it), finally robot_pos_4 (arm 1 is prepared to extend and deposit the blank on the press).

Firstly, we need to identify the marker event in RTurnCW, i.e., robot_pos_1 representing that robot_pos = 1 holds. The last communication, robot_pos', represents a continuous flow from the environment that keeps process RTurnCW of the robot arm positions informed. The modeling CSP+T process POS_1 must then engage in the event a1_finish, which reports that arm 1 is positioned, which is obviously restricted to the occurrence of the above marker event. Consequently, we have two possible cases to consider: the first represents the presence of a blank on the table that the arm 1 gripper will be able to pick it up, the second occurs when no new blank prompts on time (within the deadline T) on the table. Since we must model a process with 2 alternatives, rules 3 must be applied in order to construct it. In POS_1 the control of the robot will start turning arm1 counterclockwise (CCW) until a robot arm reaches one of the positions 2, 3, or 4.

Robot_Turn_Control ≡ RTC
RTC = start → actions(start) → RTurnCW
RTurnCW = (robot_pos_1 → a1_finish → action(RTurnCW) → POS_1
| robot_pos' → RTurnCW)
POS_1 = (I_1 (robot_pos_1, a1_finish) → actions(POS_1_CCW) → RTurnCCW
| I_2 (robot_pos_1, a1_finish) → action(Blank_timeout) → RTurnCCW)

Its associated enabling intervals are defined as follows:
I_1 (robot_pos_1, a1_finish) = [t, t+T] // indicates that the robot arm 1 has picked up a blank from the rotating table.
I_2 (robot_pos_1, a1_finish) = (t+T, ∞) // there was not blank to pick up within time T.
action(event) is a notation used to simplify the writing of syntactic terms in process specifications. It is intended to summarize all the actions associated with a transition, \textit{event} represents the condition or the process term in which it is defined. For instance, \text{action(RTurnCW)} = \text{activate_stop_turn}; t= \text{gettime()}. The syntactical term \text{RTurnCCW} represents rotation of a robot arm, which takes some time to reach the next safe state, and must be represented as a state of the STD.

\text{RTurnCCW} = ( \text{robot_pos}_2 \_ \& \_ \text{press_full} \rightarrow \text{action(POS}_2) \rightarrow \text{POS}_2 \\
| \text{robot_pos}_3 \_ \& \_ \text{a2_full} \rightarrow \text{action(POS}_3) \rightarrow \text{POS}_3 \\
| \text{robot_pos}_4 \rightarrow \text{action(POS}_4) \rightarrow \text{POS}_4 \\
| \text{robot_pos}\_?\text{robot_pos} \rightarrow \text{TTurnCCW} )

\text{POS}_2 = a2\_finish \rightarrow \text{action(POS}_2\_\text{CCW}) \rightarrow \text{RTurnCCW}

\text{POS}_3 = ( \text{a1_full} \_ \& \_ \text{a2_finish} \rightarrow \text{action(POS}_3\_\text{CCW}) \rightarrow \text{RTurnCCW} \\
| \text{a1_empty} \_ \& \_ \text{a2_finish} \rightarrow \text{action(POS}_3\_\text{CW}) \rightarrow \text{RTurnCW} )

\text{POS}_4 = a1\_finish \rightarrow \text{action(POS}_4\_\text{CW}) \rightarrow \text{RTurnCW}

The robot enters in position 2 only if there is a forged plate on the press (and it is in the down position), otherwise it goes directly to position 4, which will allow arm 1 to drop its blank on the press, thereby preventing an unnecessary turn (CCW) towards the deposit belt. The first alternative of process \text{POS}_3 represents the case in which arm 2 picked a plate from the press and there is also a blank in the other arm, and in this case it continues to turn CCW so that arm 2 can deposit the plate on the belt and then arm 1 can drop the blank on the press; the second alternative addresses the case in which there is no blank in arm 1, thus the plate in arm 2 is dropped on the belt and the robot turning state changes to clockwise (CW) in order to return both robot arms to their initial positions.

Robot Arms Control

Processes \text{RA1} and \text{RA2} model the robot arms extension, contraction and actions (on the electromagnet) needed to pick up blanks from the belts. Since we also need to model alternatives in these processes, it will again be necessary to apply rules 3.1, 3.2.

\text{Robot\_Arm1} = \text{RA1}

\text{RA1} = \text{start} \rightarrow \text{action(start)} \rightarrow \text{INIT\_A1}

\text{INIT\_A1} = \text{pos_arm}_0 \rightarrow \text{action(INIT\_A1)} \rightarrow \text{A1WLoad}

\text{A1WLoad} = a1\_ready \_ \& \_ \text{table}_\_\text{w_robot} \rightarrow \text{action(waiting_table)} \rightarrow \text{A1Extending}

\text{A1Extending} = ( a1\_empty \_ \& \_ \text{arm}_\_\text{situated} \rightarrow \text{action(picking_blank)} \rightarrow \text{A1Retracting} \\
| a1\_full \_ \& \_ \text{arm}_\_\text{situated} \rightarrow \text{action(dropping_blank)} \rightarrow \text{A1Retracting} \\
| \text{pos_arm}\_?\text{pos_arm} \rightarrow \text{A1Extending} )

\text{A1Retracting} = ( \text{pos_arm}_0 \_ \& \_ a1\_empty \rightarrow a1\_finish \rightarrow \text{blank}_\_\text{dropped} \rightarrow \text{A1WLoad} \\
| \text{pos_arm}_1 \_ \& \_ a1\_full \rightarrow a1\_finish \rightarrow \text{blank}_\_\text{picked} \rightarrow \text{A1Wunload} \\
| \text{pos_arm}\_?\text{pos_arm} \rightarrow \text{A1Retracting} )
A Transformational Approach to the Systematic Design of Real-Time Systems

When the robot arm is in position 0 and the turn control state is retracted, this means that the arm 1 is returning after having dropped the blank on the press, then the control must evolve to a state in which arm 1 is prepared to pick up the next blank (A1WLoad). If arm 1 holds a blank, it must have been taken from the rotating table, so the arm should change to position 1 (waiting for the press to be positioned), which represented by the modeling process A1WUnload. If the press is in the middle position, i.e. waiting for a new blank from arm 1, the robot turn control must receive a command (in action(waiting_press)) in order to start turning the arm CCW until the state of the arm changes to extending and the blank results dropped on the press.

The syntactical process term RA2 description is equivalent to the above RA1 one.

Robot Control

Following the ascending order of the hierarchy of schemes within the SR model, we must now model the higher abstraction scheme, the RT/SA Robot Control entity in figure 3, from its integrating processes in Figure 4. RC must offer its users a simpler communication interface than the union of the interfaces of its already modeled subprocesses RTC, RA1 and RA2 and in addition, it must be obtained by the only parallel composition of its components, without any additional re-structuring of the specification. Rule 7 will address this case in order to obtain the compounded CSP+T term named RC and its interface. This rule takes advantage of the compositionality of process terms and their algebraic properties in order to obtain new process seamlessly from its parts. All the internal events must be hidden, so that the communication interface of RC coincides with the flows in figure 3.

Robot_Control≡RC

RC = (RTC\{start, a1_finish, a1_ready, a2_finish, a2_ready, robot_pos’\}) || (RA1\{start, a1_finish, a1_ready\}) || (RA2\{start, a2_finish, a2_ready\}) || (RTP\{robot_pos’\}) || (RAP1\{pos_arm1’\}) || (RAP2\{pos_arm2’\})

Since the above control processes receive three continuous data flows (robot_pos, pos_arm1 and pos_arm2) reporting the current robot position and the horizontal positions of the arms, respectively, it has been necessary to write 3 synchronization processes to model them; therefore, by applying rule 2, we must write the following additional CSP+T process terms to complete the specification of the DTP robot control

RTP=robot_pos?x→RTP’
RTP’=(robot_pos?x→RTP’| robot_pos’!x→RTP’)
RAP1=pos_arm1?x→RAP1’
RAP1’=(pos_arm1?x→RAP1’| pos_arm1’!x→RAP1’)
RAP2=pos_arm2?x→RAP2’
RAP2’=(pos_arm2?x→RAP2’| pos_arm2’!x→RAP2’)
Table, Press, Feeding Belt, Deposit Belt Control

The same transformational approach can be applied to the Table Control Process (TC), in Figure 3, and its components: Table Turn Control, Table Lift Control and Table Load Control.

The syntactic process terms corresponding to the STDs from process Press Control (PRC) are also derived by the systematic application of the transformation rules. Press control must also use a marker event to control the amount of time the press must forge a blank to obtain a plate. The Feeding Belt Control process (FBC) must convey blanks to the rotating table. It has a sensor to check the arrival of a blank at the end of the belt, just before loading it onto the rotating table. This sensor is used to measure the size of the blank or to detect whether they overlap or are too close to be identified by the light barrier. A restricted event must be used to signal the detection of valid blank sizes. If a blank bigger than that allowed is detected, the system will stop. The sensor is used to stop the belt when the rotating table is not ready to get the blank. The Deposit Belt Control (DBC) process has a description analogous to that of the FBC process.

The Production Cell

The complete system is finally obtained by integrating all the elements plus a DS (PressStatus). This one is used to keep the press position that is received as a data flow by the robot arm control processes.

PressStatus = PS; PS = start → PS’
PS’ = (new_status?x → PS’ | current_status!x → PS)

Finally, we can integrate the entire production cell control system by parallel composition.

Production_Cell ≡ PC
PC = (FBC\{failure, start } || TC\{start, table_turn, turn_stop, table_move, move_stop } || RC\{start }|| PRC\{start }|| DBC\{start }|| PS\{start })

By applying rule 7 to hide the internal events, the system context diagram (Figure 2) can be obtained. We complete the down-up design process with the definition of the following CSP+T process term that models the whole system,

Production_Cell_Context ≡ PCC
PCC = 0. ◆ → PCC\{start, db_w_robot, plate_dropped, blank_dropped, plate_picked, table_w_robot, blank_picked, table_w_fb, blank_loaded}.

Conclusions

We have presented a derivation procedure to obtain a correct system specification from a semi-formal SA/RT system requirements specification of a given real-time system. The imprecisions and ambiguities intrinsic to SA/RT notations have been overcome in our method by using a formal description language based on the process algebra CSP+T. However, unlike other proposals that attempted to overcome the same problem by complementing SA with formal methods, our methodological
scheme uses $CSP^+T$ to provide the user with a set of patterns into which he/she can translate from $RT/SA$ entities to $CSP^+T$ processes with different semantics, which range from the most predictable interpretation (i.e. precluding nondeterministic process constructions) of analysis entities to the most flexible ones (i.e. using nondeterministic structures). The method has been defined for its easy integration in $ASE$ environments and/or formal tools based on $SA$ notations. We are currently working on the development of a formal software tool based on $CTJ$ [6] and Java, and which is capable of automated specification, verification and code generation of real-time and embedded system software for several computing platforms.

References