An Integrated Approach to Specification and Validation of Real-Time Systems

Adnan Sherif Augusto Sampaio Sérgio Cavalcante

{ams,acas,svc}@cin.ufpe.br
Federal University of Pernambuco
Center of Informatics
P.O. BOX 7851 Cidade Universitaria
50740-540 Recife - PE Brazil

This work presents an integrated approach which covers from the formal specification to the analysis and use of tools to prove properties about real-time systems. The proposed language to specify the system behaviour is Timed-CSP-Z, a combination of Timed CSP and Z. We propose a rule-based strategy for converting a Timed-CSP-Z specification to TER Nets, a high level Petri Net based formalism with time. The conversion enables us to use the CABERNET tool to analyse desired properties. As a practical case study we discuss the application of this approach to the specification and analysis of an On-board Computer of a Brazilian microsatellite.

1 Introduction

Real-time computer systems differ from general-purpose computer systems in that they introduce the time notion to the computational requirements of the system. Real-time systems must not only provide logically correct and adequate results but these results also have to be provided within a certain period of time.

First generation real-time applications were relatively simple and did not involve sophisticated algorithms or extensive computational complexity. However, things have changed in the last decades towards more complex and more safety critical applications such as aerospace navigation and control, monitoring factories and nuclear power plants, among others. The use of real-time systems in safety critical applications makes them a serious candidate for the use of formal methods concerning their specification and validation.

The choice of the language to be used in the specification of a software system is an important factor concerning the success of the entire development. The language should cover the several facets of the system requirements and it should have a suitable observation model. The model is used to study the behaviour of the system and to establish the validity of desired properties. Another important factor to be considered is the popularity of the language.

During the last years, the software engineering researchers have developed a large number of formal specification languages. However, each of these languages is suitable for expressing a particular characteristic and lacks the power of expressing others. For example, Z [Spi88] is a formal language used to define data types and to show the effect of operations on these types. But Z lacks tools to express the order in which the operations are executed [Eva94]. On the other hand, CSP [Hoa85, BHR84, Ros98] is a language suitable for showing the order of the events occurrence but lacks abstract data types and facilities to express (in a natural way) the effect of the events on data. Formalisms like temporal logic [BH81] concentrates on time aspects.

There are many proposals to extending existing languages as well as merging two or more languages and giving a new semantics to the integrated language. For example, CSP-Z, the solution proposed by Fischer [Fis96, Fis00], combines CSP and Z. CSP-Z is based on the Failure Divergence model of CSP [Ros98]. The adaptation of the model was proposed by Fischer in [Fis96, Fis00].

In this work we propose an approach to the specification and validation of real time systems. This approach was applied in the specification of the On-board Computer of the first Brazilian satellite for scientific applications (SACI-1) [SN95, dPJ95]. The language (Timed-CSP-Z) extends CSP-Z with facilities to specify real-time systems. The extension was achieved by substituting the CSP part of the language by Timed CSP [RR86, DS95] (a version of CSP capable of expressing timed behaviour). Timed-CSP-Z is based on an observation model different from the one used by CSP-Z, as explained in section 2.3

The main advantage of using formal methods in system specifications is the ability of studying a specification with the aim of assuring the presence and/or absence of properties in the system behaviour. One approach to achieving this objective is to use model checking to compare different observed behaviours of a system. Model checking is an automatic tool that helps to compare different specifications based on an observation model.

Ideally, we should use a model-checking tool like FDR [For96] to carry out such analysis, since FDR handles CSP specifications. But, unfortunately, FDR deals only with pure CSP (without time operations) and there are no tools available for the proposed language (Timed-CSP-Z).

Based on recent work in the literature related to process algebra, we adopted the solution of translating Timed-CSP-Z to a Petri Net formalism called environment relational nets (ER Net)[GMMP91]. We developed rules to convert a specification written in Timed-CSP-Z to ER Nets. We then used a tool (CABERNET) to analyse the Petri Nets resulting from converting the original specification.

The remainder of this paper is organised as follows. The proposed integrated language for the specification of real-time systems is introduced in section 2, through the specification of the Watchdog Timer (which is part of the SACI-1 satellite). In section 3 we focus on the approach to specification analysis, where we describe both a strategy to convert a Timed-CSP-Z specification into ER Nets and the use of the CABERNET tool to conduct the analysis automatically. In section 4 we summarise the results of our research and discuss topics for future work.

2 Timed-CSP-Z

Timed-CSP-Z is based on CSP-Z [Fis96], which is itself a combination of CSP and Z. CSP-Z encapsulates the process description in a specification unit containing two main blocks: a CSP block containing the CSP equations that show the sequential and concurrent behaviour of the events, and a Z block which defines a state and an operation associated with each event of the CSP part. An operation defines the precondition for an event to occur and the effect of the event occurrence on the state space. The language also permits combining the behaviour of specification units using the CSP operators.

CSP-Z is based on the Failure Divergence model of CSP [Ros98]. The adaptation of the model was proposed by Fisher in [Fis96]. In [MS00] Mota and Sampaio show the use of FDR [For96] as a model checking tool for CSP-Z.

In this section we intend to show how time aspects are added to the language in order to allow one to use the language for the specification of real-time systems. To achieve this, the CSP part of the specification is substituted by Timed CSP. A new semantic model is used as a semantic basis for the extended language (Timed-CSP-Z). In what follows we will introduce the syntax and briefly discuss the semantics of this new language.

The syntax of Timed-CSP-Z is not much different from the one of CSP-Z. The original syntax is conserved, as Timed CSP is a superset of CSP and the Z part of the language is not affected by this extension. To better express the syntax of the language we will use an example for illustration: the Watchdog Timer (WDT) process which is part of the SACI-1 On-board Computer (OBC), the first of a series of microsatellites developed by the Brazilian National Institute for Space Research (INPE). The approach was used to formally specify and analyse the satellite behaviour. A complete specification of the satellite OBC and the application of this approach to the case study can be found in [She00].

2.1 Formal Specification of the WDT

The task of the WDT is to monitor the Fault Tolerant Router (FTR) and make sure it is functioning properly. The WDT achieves this by waiting for a reset signal from the FTR within a given amount of time. If the time elapses and the reset signal is not received, the WDT sends an interrupt signal to the observed FTR and waits again for a new reset signal. If it does not receive this signal within a predefined amount of time it will consider the monitored FTR to be out of order. First the WDT will try to recover the FTR by resetting it and then returns to function normally as before. If the FTR does not respond for more than seven consecutive times then it is considered to be failed and so should be cut off from the rest of the system. If this happens then the WDT process itself will stop (behaving like the canonical deadlock process).

A process specification in Timed-CSP-Z starts with the keyword *Spec* followed by the process name.

Spec WDT_i

Observe that the process name *WDT* is indexed, which indicates that the process is parameterised by this variable. We can parameterise a process for modularity. In our case we will specify WDT_i to represent the behaviour of the *WDT* processes. When defining the complete system we will have three copies of the process, namely WDT_1, WDT_2 and WDT_3. Each occurrence of i in the process specification will be replaced by the corresponding number. The numbers in our case study represent the three different CPUs of the Satellite.

The next step is to declare the channels used by the process. The channels can be divided into external channels, prefixed by the keyword *channel*, and internal channels, prefixed by the keyword *local channel*. Channels can represent events when they have no type. If a channel is declared of an appropriate type then the channel is used for communication between processes: the information exchanged will be of the declared type. In the case of our WDT process only external event channels are declared.

channel $resetWDT_i$, $failed_i$, $WDTint_i$, $resetFTR_i$

All Timed-CSP-Z processes as previously mentioned have two main blocks. The first is the specification of the control part (in Timed CSP), and the other is the specification of the data part (in Z). The Timed CSP part starts by defining the *main* process. This process is the starting point of the Timed CSP specification.

$$main_i \;=\; (resetWDT_i \rightarrow main_i) \overset{WDTPeriod}{\rhd} Interrupt_i$$
$$Interrupt_i \;=\; WDTint_i \rightarrow (resetWDT_i \rightarrow main_i) \overset{WDTIntPeriod}{\rhd} InterruptFailed_i$$
$$InterruptFailed_i = (resetFTR_i \rightarrow main_i) \,\Box\, (failed_i \rightarrow STOP)$$

The above specification ensures that the WDT will wait for a reset signal within the time period defined between the start of the process main and the elapse of the amount of *WDTPeriod*. In case the reset is not received then the WDT shall send a *WDTint* signal to call the attention of the monitored FTR. It then waits for the reset signal for a new amount of time. If the reset is not received, the WDT sends a *resetFTR* signal re-initialising the FTR or a failed signal indicating the failure of the FTR. The decision of considering the FTR to be failed or not is determined by the Z part of the specification, which begins by defining any constants and related axioms used in the specification of the process. In the case of the WDT process we will define *WDTPeriod* and *WDTIntPeriod* as being two constants of type natural.

$$WDTPeriod : \mathbb{N}$$
$$WDTPeriod = 100$$

$$WDTIntPeriod : \mathbb{N}$$
$$WDTIntPeriod = 100$$

Next we define the state affected by each event of the process. This is defined using a Z state declaration. A special schema *Init* is also defined as the initialisation of the process state. The operational view is that it will be executed as soon as the process starts and before any activity is carried out.

State

$$FailCounter : \mathbb{N}$$
$$FailCounter \leq 7$$

Init

$$State'$$
$$FailCounter' = 0$$

Observe that the only variable in the *state* schema is a natural variable which will hold the count of failures at any given instance. It is limited by the restriction that it should hold values less or equal to 7, which is the maximum number of times the CPU can fail. The *Init* schema initialises the *FailCounter* variable with 0. The next step is to define the Z schemas relative to the Timed CSP events. The schema has the same name as the event but prefixed with the keyword com_. Each Z schema will also define the preconditions of the event. It also shows the consequence of the event execution on the data variables of the process state.

$com\_resetFTR_i$

$$\Delta State$$
$$FailCounter < 7$$
$$FailCounter' = FailCounter + 1$$

com_failed_i

$$\Xi State$$
$$FailCounter = 7$$

Observe that $com\_resetFTR_i$ is defined to have the precondition that *FileCounter* should be less than 7. The Effect of executing this event will be to increment by one the value of the counter.

$$\begin{array}{|l}
\multicolumn{1}{l}{__ com_resetWDT_i ____} \\
\Delta State \\
\hline
FailCounter' = 0 \\
\hline
\end{array}$$

$com\_WDTint_i \mathrel{\widehat=} [\Xi State]$

Observe as well that $com\_resetWDT_i$ resets the counter to zero.This event is triggered by the FTR when the condition stated by *com_failed* is satisfied. *WDTint* is declared not to affect the state of the process. These events normally represent actions or external interrupts. With this we finish the body of the WDT_i unity. All process specifications in Timed-CSP-Z terminate with the keyword *end Spec.*
end Spec

2.2 Formal Specification of the OBC

As it was previously mentioned the OBC of the SCAI-1 is composed of three CPUs. The set of the satellite application processes are partioned in the sense that each CPU will have instances of all the application processes, but only one copy of the application processes will be active in a unique CPU at a given time. To achieve this a special process none as the Fault Tolerant Router (FTR) is used. This process acts as the operating system of the OBC. As we mentioned in the previous section the process is responsible for responding to the WDT process, while its main activity is to switch messages between the application processes. So a more abstract description of the OBC will be a network of processes exchanging messages which are routed by the FTRs which in turn are monitored by the WDTs

Here we present the top level specification of the OBC system, in order to give an idea of its structure. To achieve this we will use the CSP operators to combine and join specification units. The first process defined bellow is the CPU'_i process which is the parallel composition of the WDT_i process (defined in the previous section) and the satellite Fault Tolerant Router (FTR). Next the resulting process of the first equation is executed in parallel with the rest of the application processes represented hear by the APP process (Telecommand, Telemetry .....) in order to form the complete model of the CPU. Observe that each of the application processes is a separate specification unit. In [She00] a detailed specification of each of these specification units is presented.

$$CPU'_i = FTR_i \underset{\{resetWDT_i,\, resetFTR_i,\, WDTint_i,\, failed_i\}}{\|} WDT_i$$

$$APP_i = TC_i \| TM_i \| SGC_i \| EDAC_i \| EDA_i \| SRI_i \| SDC_i \| ADC_i \| DAC_i \| HK_i \| D_i \| EMM_i$$

$$CPU_i = APP_i \underset{\{getMsg\_i,\, putMsg\_i\}}{\|} CPU'_i$$

The set of events used as subscripts of the parallel operator describes the points of synchronisation of the processes running in parallel.

Before we introduce the final equation that composes the three CPUs to form the OBC process we will define two sets that will be used in the equation.

$INTERFACE_{12} = \{failed_1, failed_2\}$
$INTERFACE_{123} = INTERFACE_{12} \cup \{failed_3\}$

Finally we define the equation that defines the complete on-board computer subsystem of the SACI-1.

$$OBC_i = (CPU_1 \underset{\text{INTERFACE}_{12}}{\|} CPU_2) \underset{\text{INTERFACE}_{123}}{\|} CPU_3$$

2.3 The Semantics of Timed-CSP-Z

The semantics of CSP-Z as defined in [Fis96] is the Failure Divergence model $\mathcal{M}_{\mathcal{FD}}$ of CSP [BR85]. An interesting aspect of the semantics of CSP-Z is that the Z part is given a semantics in $\mathcal{M}_{\mathcal{FD}}$. As CSP has already a semantics in the same model, a desired uniformity is achieved. Intuitively, a CSP-Z process can be understood as the parallel composition of its CSP part and a CSP process, say P, built from the Z part:

$$\textit{CSP-Block} \underset{\Sigma}{\|} P(\textit{Z-Block})$$

where Σ is the alphabet of the process, and the process P is parameterised by the state space of the Z block which is basically the external choice of all the events of the original CSP-Z process. These events are actually guarded by the precondition of the corresponding Z schema.

Operationally, the CSP part acts like a master process which controls the execution and the Z part as a slave process. When an event occurs in the CSP part this will synchronise with the Z part, and the associated effect in the state space will happen. A more detailed definition of the semantic model of CSP-Z can be found in [Fis96, Fis00].

Timed-CSP-Z adopts the same principles used by CSP-Z except for the use of the Timed Failure Model $\mathcal{M}_{\mathcal{TF}}$ of Timed CSP instead of the Failure Divergence model used by CSP-Z. A more detailed description about the semantic model of Timed-CSP-Z can be found in [She00].

3 The Specification Analysis: Strategy and Tools

The main advantage of using formal methods in system specifications is the ability of studying the specification with the aim of assuring the presence or the absence of properties in the system behaviour. To achieve this objective, model checking is used to compare different observed behaviours of a system. Model checking is usually performed by automatic tools, which help to compare different specifications based on an observation model.

An important observation that is studied about concurrent and parallel systems is the presence or absence of deadlock states in the system specification. This type of study will be the analysis goal of the SACI-1 OBC specification partially introduced in the previous section.

As mentioned before no analysis tools exist for Timed-CSP-Z to help in model checking. The language that gave origin to Timed-CSP-Z, CSP-Z, has no tools either. In [MS00] a method was introduced to adapt FDR (a tool used in the analysis of CSP processes) to the analysis of CSP-Z. This method uses parameterised processes to represent the state space of the specification and translates the Z part of the specification into a parametrised CSP process. This method works fine for CSP-Z as it uses the same observation models of CSP. Unfortunately, the same

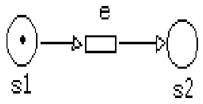

Figure 1: The PN Representation of an Event e

method cannot be adapted for Timed-CSP-Z as the observation model is different and FDR does not offer any analysis based on the timed models of CSP.

Based on recent work in the literature [MMR+98, Bal92] related to process algebra, we have adopted the solution of translating Timed-CSP-Z to a Petri Net formalism called environment relational nets (ER Net)[GMMP91] for which there exists an analysis tool. ER Nets were originally presented in [GMMP91] and consist of a high-level Petri Net where tokens are environments, i.e., functions associating values to variables. Furthermore, an action is associated with each transition describing which input tokens can participate in a transition firing and the possible tokens produced by the transition. A complete definition of ER Nets and time representation in ER NEts can be found in [GMMP91].

3.1 Converting Timed-CSP-Z to TER Nets

A method for converting a Timed-CSP-Z specification into a TER Net is proposed in this section. The conversion rules for CSP are taken from previous work related to converting CSP to Petri Nets [MMR+98, Bal92]. These conversion rules have been adapted to take into account the state space introduced by the Z part, and new rules have been added to cover the time aspects.

3.1.1 Basic Conversion Rules

Rule 0. Basic elements

The basic element of a Timed-CSP-Z specification is an event, say e, in the CSP part of the specification and a corresponding Z schema representing the Z part of the specification. Each Z schema predicate is divided into two parts: the precondition for the event to occur which depend on the value of input variables and current value of the state space of the specification, and the post-condition or the effect of the event occurrence on the state space. The translation of this basic element is shown in Figure 1. Where S1 represents the input place to the basic element and holds the data space before the event occurrence, and S2 is the output place which holds the resulting state of the data space generated by the event occurrence. The event itself is represented by the transition that connects the input places to the output places. The transition firing represents the event occurrence.

To better illustrate the rule consider the following portion of a specification:

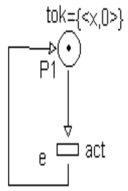

Figure 2: An Example of Converting Timed-CSP-Z to ER-Nets

Spec ex1

channel e

main $= e \rightarrow main$

State
$x : Int$

Init
$State'$
$x' = 0$

com_e
$\Delta State$
$x < 10$ $x' = x + 1$

end Spec

The example above can be translated to the ER Net shown in Figure 2.

The initial token in place P_1 represents the starting point of the specification. The *Init* schema is represented in the initialisation value of the token. The value of the token *tok* at place P_1 represents the state space before the occurrence of the event e. The resulting token represents the result of the occurrence of the event e. The event is represented by the transition e in the Petri Net. The action associated to the transition is defined from the Z schema associated to the corresponding CSP event. The action is defined by specifying the sequence of input places to the transition and the sequence of output places to the transition. Then the action behaviour is defined using a logical predicate to state the conditions on the input places and the result of the transition firing on the tokens produced in the output place.

act_e $= < P_1, P_1' > | \ P_1.x < 10$ and $P_1'.x = P_1.x + 1$

All arcs have weight 1 meaning that for each transition firing there will be only one resulting token for each place.
STOP and SKIP, which are special processes of CSP, represent a failure and a successful termination, respectively. The similarity between these two processes is that both suspend the program execution and no events occur after these processes are reached. In Petri Nets we can

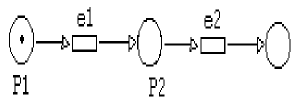

Figure 3: Sequential Composition

represent both processes as a sink place.

Rule 1. Prefixing.

Let P be a non-recursive process represented in CSP by:

$$P = e_1 \rightarrow e_2 \rightarrow STOP$$

Each event in P has its corresponding Z schema, in the Z part of the specification. Consider the following schemas.

com_e_1
$\Delta State$
$x > 10$
$x' = x + 10$

com_e_2
$\Delta State$
$x < 30$
$x' = x + 1$

The translation of each separate event can be done using rule 0. A sequential non-recursive composition of two events can be done simply by merging the output place of the first event with the input place of the next event. The result can be observed in Figure 3. The action statement for event e_1 and e_2 is described as follows:

$$\text{act\_}e_1 = < P_1, P_2 >|\ P_1.x > 10 \text{ and } P_2.x = P_1.x + 10$$
$$\text{act\_}e_2 = < P_2, P_3 >|\ P_2.x < 30 \text{ and } P_3.x = P_2.x + 1$$

For recursive processes, the difference lies in the last event output place. The same rule applies to this case, except that the last place (sink place) in the previous example is removed and the output place of the last transition is the input place of the first transition. The new action statement for events e_1 and e_2 can be described as follows:

$$\text{act\_e}_1 = < P_1, P_2 >|\ P_1.x > 10 \text{ and } P_2.x = P_1.x + 10$$
$$\text{act\_e}_2 = < P_2, P_1 >|\ P_2.x < 30 \text{ and } P_1.x = P_2.x + 1$$

Rule 2. Choice

A deterministic choice in CSP-Z is not determined only by the CSP operator, but also by the value of the preconditions of the events involved in the branching. If all preconditions are true then the choice with respect to the Z part of the specification is nondeterministic. Consider

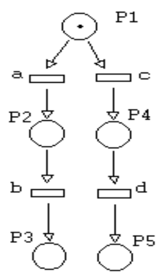

Figure 4: A Non-Recursive Deterministic Choice Operation.

the following example.

$$P = a \rightarrow b \rightarrow STOP$$
$$Q = c \rightarrow d \rightarrow STOP$$
$$main = P \,\Box\, Q$$

com_a

$\Delta State$

$x > 10$
$x' = x - 1$

com_c

$\Delta State$

$x \leq 10$
$x' = x + 1$

$com\_b \mathrel{\widehat{=}} com\_d \mathrel{\widehat{=}} [\Xi State]$

The example can be represented by the ER Net shown in Figure 4. Observe that the input place for a and c is shared. This permits that they use the same state space to validate the choice.

The actions associated with the transitions are:

act_a= $< P_1, P_2 >| \; P_1.x > 10$ and $P_2.x = P_1.x - 1$
act_b= $< P_2, P_3 >| \; P_3.x = P_2.x$
act_c= $< P_1, P_4 >| \; P_1.x \leq 10$ and $P_4.x = P_1.x + 1$
act_d= $< P_4, P_5 >| \; P_5.x = P_4.x$

A non-deterministic choice can be clearly represented by simply ignoring the precondition associated with the first event. Figure 5 clearly represents this. Consider the same previous example except that this time the nondeterministic operator of CSP is used.

The actions associated with the transitions are:

act_e1= $< P_1, P_2 >|$ true and $P_2.x = P_1.x$
act_e2= $< P_1, P_4 >|$ true and $P_4.x = P_1.x$
act_a= $< P_2, P_3 >| \; P_2.x > 10$ and $P_3.x = P_2.x - 1$
act_b= $< P_3, P_2 >| \; P_2.x = P_3.x$
act_c= $< P_4, P_5 >| \; P_4.x \leq 10$ and $P_5.x = P_4.x + 1$
act_d= $< P_5, P_4 >| \; P_4.x = P_5.x$

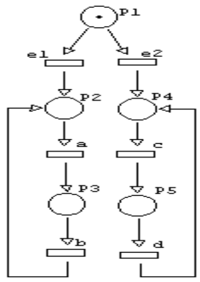

Figure 5: A Non-deterministic Recursive Choice Operation.

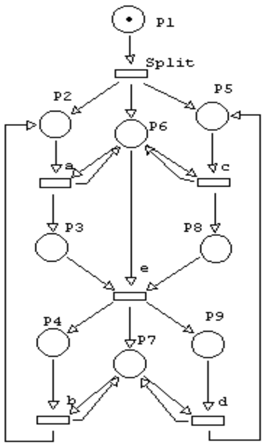

Figure 6: Synchronised Parallel Processes.

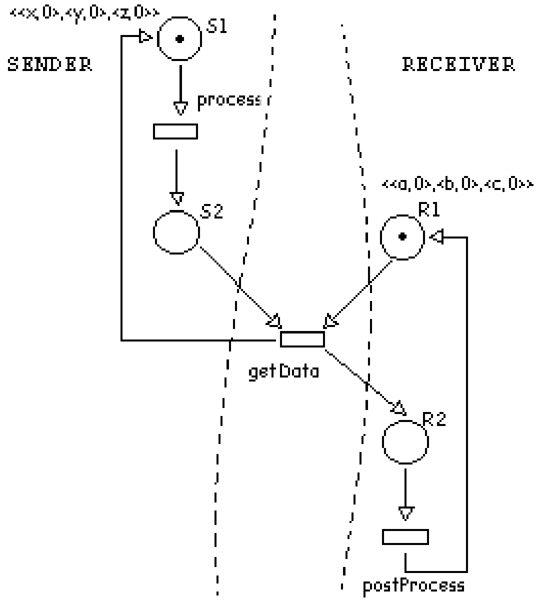

Figure 7: Synchronised Parallel Specification Units.

Rule 3. Parallel Composition.

Parallel composition is the most difficult operation to convert. This again is due to the lack of modularity of Petri Nets. Consider the case of two processes, say P and Q, that synchronise without data exchange. This happens when the processes are running in the same specification unit they share the same data space.

$$P = a \rightarrow e \rightarrow b \rightarrow P$$
$$Q = c \rightarrow e \rightarrow d \rightarrow Q$$
$$R = P \underset{\{e\}}{||} Q$$

Figure 6 presents the conversion result of the above example. Observe that the processes start in a common place, as the initialisation process is common to both of them. The transition *Split* is used to start the parallel composition. The firing of the transition *Split* results in three tokens. The first token passes control to the process P, the next passes control to process Q and the last simply carries the common data state between the running processes. This is because processes in the same specification unit share the same data space. So, even though the processes are independent of each other, and may not need to communicate, they still share the same data space. The places P_6 and P_7 act as a common storage between the processes executing in parallel.

The other type of parallel composition is the operation involving different specification units. In this case the communication is performed with data typed channels and needs to be synchronised. ER-Nets does not have any operational semantics for modelling and grouping nets into processes or models. This makes our problem more difficult to resolve. The solution is to share data spaces and to share transitions between specification units. The communication is explicitly performed at the moment of the transition firing. Instead of communication channels, the shared transition simply exchange the data values between the data spaces of the two specification units simulating a communication. Figure 7 illustrates a sender/receiver problem.

These conversion rules along with the one for the interrupt operator can be found in more details in [She00].

3.1.2 Time in ER Nets

In TER Nets, it is assumed that each environment contains a variable, called *chronos* [GMMP91], whose value is of numerical type, representing the timestamp of the token. For example, timestamps would assume natural number values, when dealing with discrete time models; on the other hand, they assume real number values in order to deal with a continuous time model. The actions associated with the transitions are responsible for producing timestamps for the tokens that are inserted into the output places based on the values of the environments of the chosen input enabling tuples. The basic idea is that the timestamp represents the time when the token was produced. In order to capture the intuitive concept of time, however, *chronos* cannot be treated as an unconstrained variable and the following axioms must be satisfied [GMMP91].

- Axiom 1: Local monotonicity. For any firing, the value of *chronos* in the environments produced by the firing cannot be less than the value of *chronos* in any environment removed by the firing.
- Axiom 2: Constraint on timestamps. For any firing $x =< enab, t, prod >$, all elements of the tuple prod have the same value of *chronos*, called the time of the firing. We denote such value as time(x).
- Axiom 3: Firing sequence monotonicity. For any firing sequence s, the times of firing should be monotonically non-decreasing with respect to their occurrence in s.

The following are the conversion rules for the timed operators.

Rule 4. Wait Operation.

The time is represented in ER-Nets by means of a variable common to all processes that can be affected by the firing of any transition in the net. For this reason, we consider the preconditions of the time passing but do not study the actual increment operation as this is considered to be automatic. However the time variable can be changed by the action of a transition to reflect the duration of an event. To illustrate the WAIT operation, consider the following example.

$P = a \rightarrow b \rightarrow P$
$main = Wait\ TimeDelay;\ P$

$$\begin{array}{|l}
TimeDelay : \mathbb{N} \\
\hline
TimeDelay = 30
\end{array}$$

$com\_b \mathrel{\widehat=} [\Xi State]$

$$\begin{array}{|l}
\textit{com_a} \\
\hline
\Delta State \\
\hline
x > 10 \\
x' = x - 1 \\
\hline
\end{array}$$

The process waits for 30 time units before performing process P. We can interpret the WAIT operation as a simple transition with a precondition based on the time the transition was armed

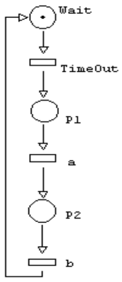

Figure 8: Example of a Wait Operation

to fire. The conversion is illustrated in Figure 8. The actions associated with the transitions are:

act_TimeOut=$< Wait, P_1 >|\ P_1.chronos \geq Wait.chronos + 30$
act_a=$< P_1, P_2 >|\ P_1.x > 10$ and $P_2.x = P_1.x - 1$
act_b=$< P_2, P_3 >|\ P_3.x = P_2.x$

Observe that the transition Wait was added to impose the time precondition.

Rule 5. TimeOut Operation.

The time out operation uses two processes. It should offer the first process for a determined period of time. If within this time the first event occurs the operation executes the first process but if the event does not occur then the process will never be executed and the control passes to the second process. Consider the following example.

$$P = a \rightarrow b \rightarrow P$$
$$Q = c \rightarrow d \rightarrow Q$$
$$main = P \overset{TimeOut}{\triangleright} Q$$

$$\begin{array}{|l} TimeOut : \mathbb{N} \\ \hline TimeOut = 30 \end{array}$$

$$com\_b \mathrel{\widehat{=}} com\_d \mathrel{\widehat{=}} [\Xi State]$$

com_a

$$\begin{array}{|l} \Delta State \\ \hline x > 10 \\ x' = x - 1 \end{array}$$

com_c

$$\begin{array}{|l} \Delta State \\ \hline x \leq 10 \\ x' = x + 1 \end{array}$$

The conversion of the example above is similar to the conversion of the choice operator. In this case, as the processes are recursive we need to create a new event to show the difference between the first event, when it occurs in the timeout operation, and the normal behaviour of the event. The conversion can be seen in Figure 9. The actions associated to the net are:

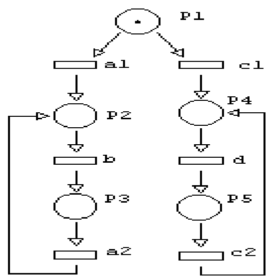

Figure 9: Example of a Timeout Operation.

act_a$_1$=$< P_1, P_2 >|\ P_2.chronos \leq P_1.chronos + 30$ and $P_1.x > 10$ and $P_2.x = P_1.x$
act_c$_1$=$< P_1, P_4 >|\ P_4.chronos \geq P_1.chronos + 30$ and $P_1.x \leq 10$ and $P_4.x = P_1.x$
act_a$_2$=$< P_2, P_3 >|\ P_2.x > 10$ and $P_3.x = P_2.x - 1$
act_b=$< P_3, P_2 >|\ P_2.x = P_3.x$
act_c$_2$=$< P_4, P_5 >|\ P_4.x \leq 10$ and $P_5.x = P_4.x + 1$
act_d=$< P_5, P_4 >|\ P_4.x = P_5.x$

Observe that the transitions a_1 and c_1 are used in order to add a time precondition to the processes. The action associated with each transition imposes the timing restrictions. That is a_1 only executes within 30 time units from initialisation while c_1 only executes after 30 time units from the initialisation, but never after a_1. Observe as well that the time intervals used in the precondition overlap. If the time is within the overlapping period then the choice is nondeterministic as imposed by the description of the Timeout operation in Timed CSP.

Rule 6. Timed Interrupt Operation.

The timed interrupt is similar to the interrupt operation, except that a time precondition is added to the start of the interrupting process. These preconditions normally represent the time at which the first process should terminate and the second should begin. The actions of the interrupted process should have a way to avoid their execution after been interrupted. Consider the following example. We omit the Z schemas associated with the events since they are the same as the in the previous example.

$P = a \rightarrow b \rightarrow P$
$Q = c \rightarrow d \rightarrow Q$
$main = P_{\ IntTime} \swarrow Q$

The conversion of the example above is shown in Figure 10. The following represents the actions associated to the transitions.

act_out$_1$=$<< P_1, P_3, P_4 >|\ P_4.chronos \geq P_3.chronos + 30$
act_out$_2$=$<< P_2, P_3 >, P_4 >|\ P_4.chronos \geq P_3.chronos + 30$
act_a=$< P_1, P_2 >|\ P_1.x > 10$ and $P_2.x = P_1.x - 1$
act_b=$< P_2, P_1 >|\ P_1.x = P_2.x$
act_c=¡P$_4$, $P_5 >|\ P_4.x \leq 10$ and $P_5.x = P_4.x + 1$
act_d=$< P_5, P_4 >|\ P_4.x = P_5.x$

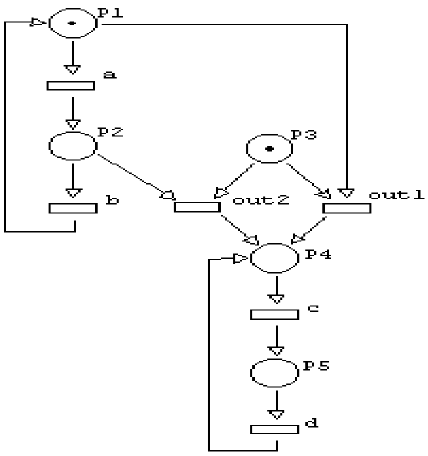

Figure 10: Example of a Timed Interrupt Operation.

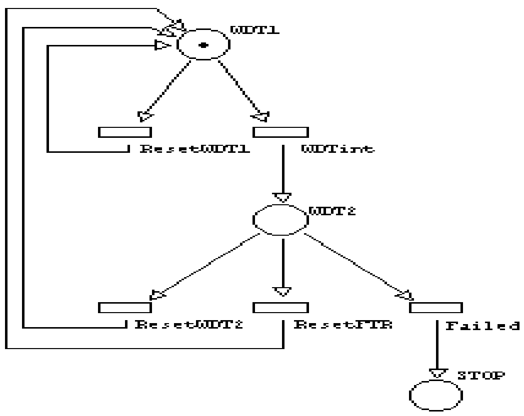

Figure 11: The PN Representation of the WDT process

The transitions a and b in this case are weak in the sense that they can only fire if there are no strong (normal) transitions enabled to fire. This condition avoids the transitions a and b from firing after the time out period.

3.2 An Analysis Example

As a conversion example we present the application of the conversion rules to the WDT process of the SACI-1 OBC. Figure 11 shows the Petri Net resulting from applying the conversion rules to the WDT specification previously described. The event $resetWDT$ is converted into two transitions as for each occurrence of an event in the CSP part a transition is added to the net.

The net starts by determining whether or not to fire the transition $ResetWDT1$ that represents the reception of the event $resetWDT$ before its deadline. If not, the transition $WDTint$ representing the timeout operation is fired. The actions associated to the transitions are:

$act\_ResetWDT1 =< WDT_1, WDT_1' >| \ WDT_1'.chronos \leq WDT_1.chronos + WDTPeriod$
$\quad and WDT_1'.FailCounter = WDT_1.FailCounter$
$act\_WDTint =< WDT_1, WDT_2 >| \ WDT_2.chronos > WDT_1.chronos + WDTPeriod$
$\quad and WDT_2.FailCounter = WDT_1.FailCounter$

If the process is not reset within the predetermined amount of time then the process sends an interrupt signal represented in our net by the transition *WDTint*. Afterwards the process makes the choice of waiting for another reset signal (represented by the net as the transition *ResetWDT*2) or whether it should reset the CPU (represented by the transition *ResetFTR*), or still whether it should consider the CPU to have failed (represented in the net by the transition *Failed*). Once the process is considered failed it terminates. This is represented by the sink place *STOP*. The actions associated with the transitions are:

$act\_ResetWDT2 =< WDT_2, WDT_1 >| \ WDT_1.chronos \leq WDT_2.chronos + WDTIntPeriod$
$\quad and WDT_1.FailCounter = WDT_2.FailCounter$
act_ResetFTR = $< WDT_2, WDT_1 >| \ WDT_1.chronos > WDT_2.chronos + WDTIntPeriod$
$\quad and WDT_2.FailCounter < 7$
$\quad and WDT_1.FailCounter = WDT_2.FailCounter + 1$
$act\_Failed =< WDT_2, STOP >| \ STOP.chronos > WDT_2.chronos + WDTIntPeriod$
$\quad and WDT_2.FailCounter = 7$
and $STOP.FailCounter = WDT_2.FailCounter$

The above example is a subset of the complete specification of the SACI-1 OBC. In [She00] a more detailed description of the conversion rules and their application to the rest of the SACI-1 OBC processes can be found.

The resulting nets serve as input to the CABERNET analysis tool. The tool uses Cab nets, a typed version of ER Nets where the actions are converted to C++ statements and the tokens are described using C++ classes. One of the problems found using CABERNET is the absence of abstract data types. During the specification we make use of many abstract data types to eliminate the need of details related to their implementation. As part of the conversion process, we need to carry out some data refinement to implement these abstract types as more concrete data types (available in C++).

Another problem found using CABERNET involves the specification size. The tool was unable to handle large nets and so some Petri Nets reduction rules were used and the specification was broken in parts to be analysed.

By applying the tool analyser upon the satellite OBC specification we observed that some subsystems of the satellite may lead to a deadlock state based upon the values of the systems timing variables. The analysis process carried out should serve as an important tool to the designer to predict consequences of any changes on the time requirements. See [She00] for further details.

4 Conclusion

We presented in this paper an approach to the specification and validation of real-time systems and discussed the application of this approach to the specification and analysis of an industrial case study: the microsatellite SACI-1 On-Board Computer (OBC). The proposed approach suggests the use of Timed-CSP-Z for specification and a Petri Net based formalism for analysis (specification validation). Timed-CSP-Z allows us to capture the several facets of a real-time system in an integrated way: the control flow, possibly concurrent and distributed (in CSP), the data space in terms of abstract data types (in Z) and the time aspects (in Timed-CSP). In order to validate the specification, a strategy was suggested to allow a systematic conversion of Timed-CSP-Z specifications into TER Nets. Finally we single out the use of the CABERNET tool to conduct the specification analysis.

An important result of this work has been its practical application: the formal specification of the timing aspects of the satellite using Timed-CSP-Z, the analysis of the effect of changing the timing parameters on the behaviour of the satellite, and the presence or the absence of deadlock in such cases.

The application of formal methods to a real and complex system offers various potential benefits. Although the specification covers only a subset of the SACI-1 OBC we hope that it can be useful in the following ways.

- A precise documentation of the functionality of critical parts of the satellite can be used as a reference for the group responsible for developing the system. It can also be used as a reference for the support group during the maintenance phase of the project, or the development of new projects based on the SACI-1.
- The formal description is a solid basis for the verification of the satisfaction of desired properties and the absence of undesirable properties in the system specification. Especially because the system cannot run in its natural environment for testing, the tests need to cover all the possible combinations and under the different possible conditions. Simulating the occurrence of errors and tracing the specification behaviour with the CABERNET might be very helpful for achieving this goal.

The specification of the On-Board Computer (OBC) of the SACI-1 briefly illustrated here (and more completely in [She00]) is based on an untimed specification which was developed using CSP-Z [Mot97]. We have not only developed a more detailed specification of each component of the OBC but also dealt with all the time aspects.

During the development of this work we observed that a similar work was carried out by Dong and Mahony [MD98], where they introduce a new language based on the combination of Timed CSP and Object-Z. The new language is named Timed Communicating Object-Z, which models objects as processes and gives the Object-Z objects predicates and operations based on the CSP syntax. This has the disadvantage of not having a clear separation between the different aspects of the process where the behaviour of the process is incorporated into the data state of the system, while in our approach the control state is totally independent of the data state and can be studied separately; this makes the specification more readable as well. Furthermore, to our knowledge, no significant specification of a case study has been developed using Timed Communication Object-Z. Concerning analysis, no approach has been suggested. In our work we have dealt with these aspects in an integrated way.

Another similar approach, Real-Time Z (RT-Z) was introduced in [Suh99]. Although the work is based on the same basic elements, the difference lies first in the syntax to integrate the Z part of the specification with the CSP part of the specification. In this approach, the use of special constructs tends to make it difficult to distinguish where the data part of the process starts and where it terminates. Another difference between this approach and ours is the use of bi-directional schemes and events to simulate the start of an operation and the end of the same operation. This approach makes it possible to describe actions and considers time operations on the start and end of a particular action. No strategy or tools were mentioned to be used to analyse specifications written in RT-Z, as well as no significant specification of a case study seems to have been developed using RT-Z.

Although we believe our approach is promising, it is important to justify the use of different formalisms for the specification and analysis, since this has required a conversion between them. The use of Petri Nets for analysis was a consequence of the lack of tools to validate the Timed-CSP-Z specification. The positive side is that the conversion was systematised by explicit conversion rules (although those rules have not been formally verified).

One might also claim that it would perhaps be more practical to have started with Petri Nets from the very beginning. Although this would avoid the need for a conversion between formalisms, we would lose the more abstract and modular presentation allowed by a Timed-CSP-Z specification.

To make the approach of more practical use, an alternative is to automate the proposed conversion strategy. From a more theoretical point of view it would also be necessary to prove the correctness of such conversion rules, based on the formal semantics of the two formalisms. Another alternative is to avoid the conversion and build an analysis tool for Timed-CSP-Z, possibly extending FDR [For96]. The advantage of developing an analysis tool for Timed-CSP-Z is that the conversion to a different formalism and all the related inconveniences are avoided.

On the other hand, converting to Petri Nets (as we have done here) might have some potential benefits towards an implementation of the specification. The converted specification can be introduced into a Petri Net based scheduler, for example into the scheduler introduced in [Cav97], to obtain the possible scheduling information regarding the system processes.

Acknowledgements

We would like to thank Alexandre Mota for his invaluable help with information regarding the SACI-1 OBC, and for the discussions related to this research.

References

[Bal92] G. Balbo. Performance Issues in Parallel Programming. In *13th International Conference on Application and Theory pf Petri Nets*, volume 616, pages 1–23, Sheffield, UK, 1992. Lecture Notes in Computer Science.

[BH81] A. Bernstein and P.K. Harter. Proving Real-time Properties of Programs with Temporal Logic. In *Proceedings 8thSymposium on Operating System Principles, ACM SIGOPS*, pages 1–11, 1981.

[BHR84] S. D. Brookes, C. A. R. Hoare, and A. W. Roscoe. A Theory of Communicating Sequential Processes. *Journal of the Association for Computing Machinery*, 31(3):560–599, July 1984.

[BR85] S. D. Brookes and A. W. Roscoe. An improved failures model for communicating processes. In *Lecture Notes on Computer Science*, volume 197, pages 281–305, 1985.

[Cav97] S. V. Cavalcante. *A Hardware-Software Co-Design System for Embedded Real-Time Applications*. PhD thesis, Department of Electrical and Electronic Engineering, University of Newcastle upon Tyne, England, 1997.

[dPJ95] A. R. de Paula Jr. Fault Tolerance Aspects of the SACI-1. *VI Simpósio de Computadores Tolerantes a Falhas*, 1995.

[DS95] J. Davies and S. Schneider. A brief history of timed csp. *Theoretical Computer Science*, 138(2):243–271, 1995.

[Eva94] A. S. Evans. Visualising Concurrent Z Specifications. In J. P. Bowen and J. A. Hall, editor, *Z User Workshop*, pages 269–281, Cambridge, June 1994. Proceedings of the 8th Z User Meeting, Springer-Verlag Workshop in Computing.

[Fis96] C. Fischer. Combining CSP and Z. Technical report, University of Oldenburg, 1996.

[Fis00] C. Fischer. *Combination and implementation of processes and data: from csp-oz to java*. PhD thesis, University of Oldenburg, 2000.

[For96] Formal Systems (Europe) Ltd. *FDR: User Manual and Tutorial, version 2.01*, August 1996.

[GMMP91] C. Ghezzi, D. Mandrioli, S. Morasca, and M. Pezze. A Unified High-level Petri Net Formalism for Time-Critical Systems. *IEEE Transactions on Software Engineering*, 17(2):160–172, 1991.

[Hoa85] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.

[MD98] B. Mahony and J. Song Dong. Blending Object-Z and Timed CSP: An introduction to TCOZ. In *Proceedings of the 1998 Internaltional Conference on Software Engineering*, pages 95–104, 1998.

[MMR+98] A. Mazzeo, N. Mazzocca, S. Russo, C. Savy, and V. Vittorini. Formal Specification of Concurrent Systems: A Structured Approach. *The Computer Journal*, 41(3):145–162, 1998.

[Mot97] A. Mota. Formalização e Análise do SACI-1 em CSP-Z. Master's thesis, Departamento de Informática, Centro de Ciências Exatas e da Natureza, UFPE, 1997.

[MS00] A. Mota and A. Sampaio. Model-Checking CSP-Z, Strategy, Tool Support and Industrial Application. *to appear in Science of Computer Programing*, 2000.

[Ros98] A. W. Roscoe. *The Theory and Practice of Concurrency*. Prentice-Hall International, 1998.

[RR86] G. M. Reed and A. W. Roscoe. A timed model for communication sequential processes. In *Proceedings of ICALP '86*, volume 226. Lecture Notes in Computer Science, 1986.

[She00] A. Sherif. Formal Specification and Validation of Real-Time Systems. Master's thesis, Departamento de Informática, Centro de Ciências Exatas e da Natureza, UFPE, 2000. http://www.di.ufpe.br/˜ams/thesis.html.

[SN95] M. E. Saturno and J. B. Neto. Software Requirement Specification for the OBC/SACI-1 Application Programs. Technical report, Instituto Nacional de Pesquisas Espaciais - INPE, 1995.

[Spi88] J. M. Spivey. *Understanding Z: a specification language and its formal semantics*. Cambridge Tracts in Theoretical Computer Science 3, 1988.

[Suh99] C. Suhl. RT-Z: An Integration of Z and timed CSP. In *Proceedings of the 1st Internaltional Conference on Integrated Formal Methods*, 1999.