

## Structured formal development in VSE II: the Robertino case study

Georg Rock, Werner Stephan, Andreas Wolpers, Michael Balsler, Wolfgang Reif, Stefan Scheer

### Angaben zur Veröffentlichung / Publication details:

Rock, Georg, Werner Stephan, Andreas Wolpers, Michael Balsler, Wolfgang Reif, and Stefan Scheer. 1999. "Structured formal development in VSE II: the Robertino case study." In *Sicherheit und Zuverlässigkeit software-basierter Systeme: Beiträge zum GI-Workshop in Bad Honnef, 3.-5. Mai 1999*, edited by Francesca Saglietti and Wolfgang Goerigk, 138–52. Garching: ISTec - Institut für Sicherheitstechnologie.

# Structured Formal Development in VSE II: The Robertino Case Study

Georg Rock<sup>1</sup>, Werner Stephan<sup>1</sup>, Andreas Wolpers<sup>1</sup>,  
Michael Balsler<sup>2</sup>, Wolfgang Reif<sup>2</sup>, and  
Stefan Scheer<sup>3</sup>

<sup>1</sup> German Research Center for Artificial Intelligence GmbH, Stuhlsatzenhausweg 3,  
66123 Saarbrücken, Germany, email: rock@dfki.de

<sup>2</sup> Abt. Programmiermethodik, Universität Ulm, D-89069 Ulm, Germany, email:  
<name>@informatik.uni-ulm.de

<sup>3</sup> European Commission, Joint Research Centre (JRC), Ispra (VA), Italy, email:  
Stefan.Scheer@jrc.it

## 1 Introduction

We describe an application of the *Verification Support Environment (VSE)* system to a real life industrial case study: Construction of a formal specification of a large, reactive and distributed control system for controlling robots in safety critical environments. This case study was used to evaluate the VSE methodology.

The *Verification Support Environment*, which was developed on behalf of the German *Bundesamt für Sicherheit in der Informationstechnik*, is a tool to formally specify and verify complex systems. It provides means to structure specifications and supports the development process from the specification of a system to the automatic generation of code. Formal developments following the VSE method are stored and maintained in an administration system that guides the user and maintains a consistent state of the development. An integrated deduction system provides proof support for the deduction problems arising during the development process.

The paper is organized as follows: First we present the robot, which was the objective of the case study and we give a short overview of the enhanced VSE system and methodology. After this introducing part a system overview of the robot and parts of the safety model are presented. The formal specification is given in chapter 6. The last chapter presents the results of the case study and gives a short outlook.

## 2 Robertino

### 2.1 Robot System

The ROBERTINO heavy robotics facility (see Figure 1) is a big 4-degree-of-freedom and 1/3 scale gantry robot, built for the design validation of the Remote

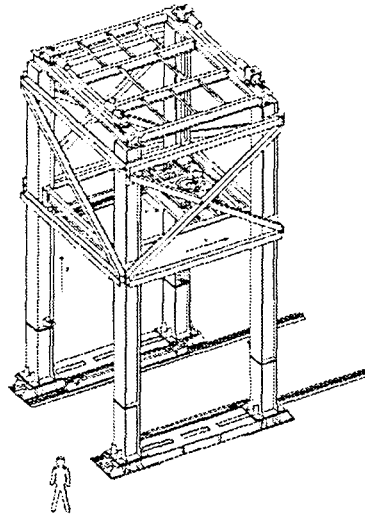


Fig. 1. Robertino prototype

Handling devices, tooling and procedures in thermonuclear fusion reactors [11]. Moving and substituting of the segments (*blankets*) of each sector of the fusion reactor requires special technology to be developed in order to carry those big (up to 15m) and heavy (up to 135 tons) blankets. In addition, access to the fusion reactor is quite limited which requires a very exact positioning. Basic safety case of the overall facility is not to disrupt the reactor mantle while the robot is operating. In other terms, the *motion control* must be sensitive enough to avoid joints and structure damages due to induced reactions caused by inertia forces and vibrations. Major design characteristics of Robertino are the following:

- It is a Cartesian robot with 4 degrees of freedom ( $x, y, z, w$  axes); the  $w$ -axis is for the gripper used to grasp the segments. The gripper is placed on a bridge which operates on the  $y$ -axis. The bridge itself is placed on a platform which operates on the  $x$ -axis. The platform itself can be moved up and down ( $z$ -axis) and it is fixed on four columns.
- It is able to move high loads: up to 6.5 tons within a range of  $2.2\text{m} \times 3\text{m} \times 6.5\text{m}$ .
- It can move and position the segments with an accuracy of better than 0.1mm and better than 0.01deg, respectively.
- It can move in  $x, y$ , and  $z$ -dimension with a maximum speed of 20mm/s. The gripper can turn with maximally 2deg/s.
- It can accelerate/decelerate in the range of 0.1 to  $5\text{mm}/\text{sec}^2$  for the  $x, y, z$  axes, and in the range of 0.1 to  $2\text{deg}/\text{sec}^2$  for the  $w$  axis.

The four logical axes correspond to nine physical axes ( $x_1, x_2, y_1, y_2, z_1, z_2, z_3, z_4, w$ ) each with its own motor. Therefore appropriate projections between physical and logical axes have to be made throughout the system model.

Robertino has a workspace due to the above mentioned range: the  $x, y, z$  limits plus the area reached by the end-effector define the global workspace. The reference point is located at the bottom of the  $z$ -direction. Extra-travel zones are allowed for special procedures as axes homing, motors setup mode or extra-travel conditions recovery.

## 2.2 Robertino Control System

The Robertino Control System (RCS) is a kernel system responsible for the axes movements and axes control. Due to the nature of the tasks of the RCS it is safety critical. RCS is a generic multi-tasking software system designed for robot applications. It is especially configured to ease functional expansions and integration of system modules for managing additional sensors and for augmenting the number of degrees of freedom.

One of the major tasks of RCS is to synchronize the motors for each of the  $x, y$  and  $z$  logical axes. This is achieved by ensuring that the error of the motor position with respect to the axis position, called motor travel offset, does not overtake a predefined threshold. If a threshold is overtaken, all axes must be stopped by an emergency stop procedure.

Since the response time of the motors plus actuators is about 5msec, the execution rate of each motor control loop is performed at most every 5msec.

Robertino can be run in various modes according to normal or special (resetting, homing, single axis movements etc.) tasks. Axes movements require an extra enabling/disabling condition in order to control better the single behaviour of Robertino. Emergency stop requests are due to emergency stop conditions which can be due to RCS external or internal causes. RCS external causes among others can be

- Emergency stop button pressed on the Control Panel or the Power & Control Unit
- Triggering of extra-travel sensors.

RCS internal causes can be due to software internal errors.

In RCS data requests concern general and detailed information about

- Axes (position, acceleration, status etc.)
- Global status (motion status: running, halting etc., RCS mode)
- Motors status for each axis.

The Robertino Control System has been developed following the ESA Software Engineering Standards; major argument for this was also to obtain RCS software characteristics such as reliability, maintainability, ability to upgrade, and testability.

### 2.3 Future Enhancements

Spin-offs of Robertino technology affect:

- Extension of its applicability: Robertino technology is applied to industries with similar requirements (moving of heavy loads, positioning accuracy) to those of Robertino: shipyards, steel industry.
- Generalisation of its control system: RCS has been chosen for technology transfer to obtain a commercial generalised software system for industrial robots. Recent developments resulted in having available a commercial software package, called GENERIS. It is entirely platform-independent and scalable, and it can easily be tailored and installed for any robotics application (including robotics equipment, hardware control platform, capabilities and operators interfaces) through a set of configuration, tuning and programming tools.

### 2.4 Motivation of Formal Model in VSE

Parts of the Robertino Control System were specified in an earlier case study with VSE I [12]. The experience made here induced us to specify the RCS using VSE II in order to evaluate the new features of the VSE II system. It was an excellent case study (accompanying the VSE II development) for the following reasons:

- Distributed system: The RCS software actually runs on two independent processors thus providing the development team with a variety of interesting aspects concerning coordination of tasks.
- Real-time aspects: As the RCS software has to deliver and to work with results that are based on extremely accurate measurements and calculations while the robot is moving, it is of high importance to investigate on real-time aspects. These aspects had to be considered during the VSE-II development and thus the case study was of great value.
- Safety-critical aspects: The behaviour of the robot during its moves may effect safety issues which, partly, are of high criticality. On the other side, the creation of a safety model within VSE is crucial and will play a fundamental part when executing the proofs.

## 3 VSE–Verification Support Environment

The Verification Support Environment (VSE) system is a tool for the formal development of software systems. It consists of:

- A basic system for editing and type checking specifications and implementations written in the specification language VSE-SL.
- A facility to display the development structure.
- A theorem prover for treating the proof obligations arising from development steps as for example refinements.

- A central database to store all aspects of the development, including proofs.
- Automatic management of dependencies between development steps.

Compared to VSE I [3,4], which was based on a simple, non-compositional approach for state based systems, VSE II [5] is enlarged and extended with respect to comprehensive methods in order to deal with *distributed* and *concurrent systems* [9] and with respect to an even more efficient and uniform proof support which makes use of implicit structuring of the arising proof obligations. The basic formalism used in VSE II is close to TLA (Temporal Logic of Actions) as for example presented in [6]. Also a refined *correctness management* allows for an evolutionary software development without proving everything from scratch again.

VSE is based on a methodology to use the structure of a given specification (e.g. parameterisation, actualisation, enrichment, or modules) to distribute also the deductive reasoning into local theories [10] [8]. Each theory is considered as an encapsulated unit, which consists of its local signature and axioms. Relations between different theories, as they are given by the model-theoretic structure of the specification, are represented by different links between theories. Each theory maintains its own set of consequences or lemmata obtained by using local axioms and other formulas included from linked theories.

This method of a structured specification *and* verification is reflected in the central data structure of a *development graph* (see Figure 3), the nodes of which correspond to the units mentioned above. It also provides a graphical interface for the system under development. Different types of specifications are displayed as different types of nodes, e.g. abstract data types as hexagons, while the relations between the nodes are displayed as links in the development graph.

### 3.1 Specifications of Concurrent Systems

In the Robertino case study the state based specification parts of the Robertino Control System (RCS) take an important role. The system architecture shown in Figure 2 indicates the parts of the control system. In the specification of these units elementary specifications and different structuring operators for state based systems are used to manage the complexity of the development.

### 3.2 Elementary Specifications

For the specification of state transition systems a specification language close to TLA [1,6,2] is used. In addition to the theory of compositional development presented in [2], which covers the composition of systems using input and output variables, *shared variables* are supported by the structuring operators in VSE II. The form of a specification of a component, also discussed in [2], is

$$\exists x_1, \dots, x_n. (\text{INIT} \wedge \square[\text{POSSIBLE-STEPS}]_{\bar{v}} \wedge \text{FAIR}),$$

where POSSIBLE-STEPS are the steps made by the system,  $\bar{v}$  is the stuttering index, which contains (a part of) the (flexible) variables of the system, and FAIR stands for the fairness requirements of the system.

### 3.3 Structuring of Specifications

VSE II provides two operators to structure state-based specifications: **combine** and **include**. We first focus on the **combine**-operator which models the concurrent execution of components. As can be seen in Figure 2, the NAC (Numerical Axes Control) state machine consists mainly of the (concurrent) composition of the state machines Interpolator, Control Panel Interface and the Field Interface. Concurrency is modeled by considering all possible *interleavings* of actions of the combined systems. Basically a behavior  $\sigma$ , which represents a sequence of states of the specified system, is a behavior of the combined system if and only if it is a behavior of every component of the system. However, in order to model the concurrent execution of, say  $S_1$  and  $S_2$ , by conjunction, we have to allow environment steps in the (local) specifications of  $S_1$  and  $S_2$ . In [2] environment steps are modeled by stuttering. This technique only works for communication by input-output variables, not in connection with shared variables. A more general approach (see [10,5]) is to associate a “color” with each component and to mark each step in a behavior by the color of the component which has done the step.

The second operator for the composition of systems, say  $S$  and  $S'$ , is **include**.  $S$  **include**  $S'$ , represents a mechanism to provide certain functionalities which are often used in the specification of systems without re-specifying them. It allows for a hierarchical composition of state machines. The semantics of  $S$  **include**  $S'$  is the conjunction of  $S$  and  $S'$ . The steps of  $S'$  in the composed system are not left open but the actions of the system  $S'$  occur now in the system  $S$ . In this way the externally visible behavior of  $S$  is also determined by the system  $S'$ . This composition style can end up in an inconsistent system. So proof obligations arise which have to be proved in order to assure the consistency of the composition.

### 3.4 Structured Deduction

Structuring specifications as described above supports readability and makes it easier to edit specifications in that the user might use local notions. However, the system exploits structure beyond this purely syntactical level. Components of a combined system can be viewed as systems in their own right where certain parts can be observed from outside while most of the inner structure, including the flow of control and local program variables are hidden.

In particular we can prove properties of a combined system like *RCS* in a modular way. This means that we attach local *lemma bases* to components where local proofs are conducted and stored. Exchange of information between lemma bases is on demand. This approach has two main advantages:

- The given structure of the specification is used to reduce the search space in the sense that large parts of the overall system are not visible.
- Storing proofs local to certain lemma bases and making the export and import of information (between lemma bases) explicit supports the *revision* process.

## 4 RCS Architecture

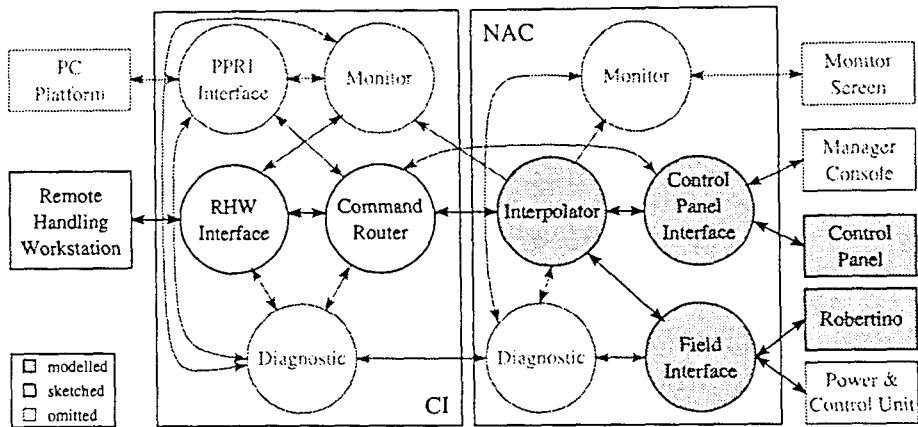


Fig. 2. RCS architecture

The system architecture is depicted in Figure 2. The Robertino Control System consists of ten processes running on two processors. A first processor CI (Communication Interface) is dedicated to communication with controlling computers, a second processor NAC (Numerical Axes Control) controls the robot. In the formal model the diagnostic and monitoring processes were omitted, as they are not safety critical. The main functionality for controlling Robertino is distributed amongst the two processes *Interpolator* and *Field Interface*. A third process *Control Panel Interface* evaluates input from a Control Panel. Because the interface to the Control Panel is safety critical and nontrivial, it was also included into the formal model. The interface to a remote handling workstation is simple, thus construction of a formal model was only sketched. A similar connection to a PC was omitted. In the formal model, commands issued by the workstation or the PC are directly passed to the *Interpolator*.

Overall, the formal model in VSE captures the main safety critical functionality of RCS to a large extent. In the following, the functionality of the three main processes as well as communication between processes are informally described in more detail.

### 4.1 Functionality

A number of buttons and joysticks are located on the Control Panel. Input from the Control Panel is parsed by the Control Panel Interface and appropriate messages are sent to the *Interpolator*. Especially *move* commands are

issued, if joysticks are pressed. These move commands are sent from the Control Panel Interface to the Interpolator. The Interpolator is a large component which includes several operating modes of the Robertino Control Software. The Interpolator computes macro targets for the movement of Robertino every 40 ms. These move commands are sent to the Field Interface which is the interface between the Interpolator and the logical control of the physical axes (indicated in Figure 2 by the Robertino box). The Field Interface provides information about the current sensor data of Robertino to the Interpolator and it computes 8 micro targets for one macro target to realize a continuous movement of the robot. Certain information about the internal state of the software is passed on to the Control Panel Interface. This information is then displayed via several lights on the Control Panel.

In case of an emergency, the emergency stop button on the Control Panel is pressed. This input is internally processed with high priority leading to an immediate, but smooth deceleration of Robertino.

## 4.2 Communication

As can be seen in Figure 2 the Interpolator is a central component of the NAC. It communicates mainly with the Control Panel Interface and the Field Interface. Having a closer look at the specification of the communication with the Field Interface we can identify two communication concepts:

1. Message queues (with priorities)
2. Input, output or shared variables (with semaphores)

The move commands from the Interpolator are inserted into the message queue of the Field Interface with respect to their priorities. The Field Interface removes the commands from the queue (the queue is a shared variable) and executes them in order.

The information needed by the Interpolator to compute the next target is provided by the Field Interface by procedure calls. This communication relies on input, output variables and on a simple synchronisation mechanism (between the Interpolator and the Field Interface).

A special communication command is the *interrupt*. The interrupt conditions are checked for at the beginning of each macro cycle in the Interpolator. In this way there is only a negligible time delay until the system reacts to an interrupt command.

## 5 Safety Model

A safety model of a system is an abstract specification of desired features of the system. In this context it is important not to specify the whole system again in the safety model but to extract the critical properties and to describe only these. Furthermore the safety model can contain liveness properties, but in case of RCS mainly safety related aspects occur. Some of the safety requirements which have to be satisfied by the Robertino Control System are:

- If Robertino gets a move command, then it will move into the right direction and will usually arrive.
- Robertino will usually be inside its workspace limits.
- There are no deadlocks in the communication between the system components of the robot.
- The given acceleration and deceleration limits will be obeyed.

As an example of a formalisation of a safety property we have a closer look at the workspace limits property. There we have to formalise the workspace, the halting, the position of Robertino and what it means that Robertino is “usually” inside its workspace. The formula which expresses the desired property is:

$$\begin{aligned} &\Box( pos_{in} \in Workspace \\ &\quad \vee( pos_{in} \in ET\_zone \wedge (now = t_0 \Rightarrow \Diamond(now < t_0 + 0.5 \wedge vel_{in} = 0))) \\ &\quad \vee(at(pos_{in}, limit\_switch) \wedge Enabled(emergency.ack) \wedge \\ &\quad \quad \Diamond(emergency.ack))) \end{aligned}$$

The box in front of the formula tells us that the described property should always hold, i.e. in every state of a behavior of the system. The rest of the formula is a disjunction of three cases:

- The first part says that Robertino, i.e. the current position of Robertino, is in the normal workspace. The next two formulas of the disjunction represent the formalization of the interpretation of “usually”.
- The second part says that if the current position of Robertino is in the *ET\_zone* (Extra Travel zone) then Robertino will halt within 0.5 time units.
- The last part is concerned with the case that Robertino is exactly at the limit switches of the workspace. In this case the emergency is enabled and will occur sometime in the future.

Specifying all safety properties in this way results in the specification of the safety model of Robertino.

## 6 Formal Specification

Specifying Robertino Control System with VSE II results in a structured formal specification. Processes are specified as temporal components which are combined using the structuring operators of VSE. Data types are specified algebraically. In the following an overview of the specification concepts is given. Since real time is important in this case study, specifying real time is also explained.

### 6.1 Specification Structure

Figure 3 contains parts of the specification of Robertino Control System. In VSE the structure of a specification is displayed graphically. Because of its size, the specification of RCS is distributed amongst several development graphs. Top level graph *RCS* is displayed on the top of Figure 3. Here, a node *NAC* combines

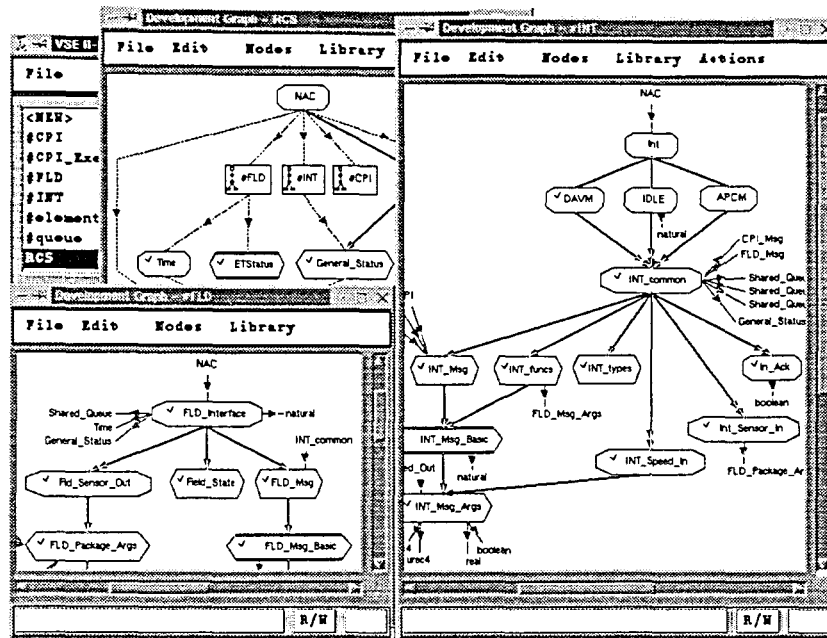


Fig. 3. Development graph

three processes *FLD*, *INT*, and *CPI* which are themselves specified in sub graphs. The development graphs of the Interpolator and of the Field Interface are also shown.

It is to be noted, that the overall structure of the formal specification is closely related to the system architecture of Figure 2. While developing VSE, special attention was turned to appropriate structuring operators. The operators allow to break down large specifications into modular components. This has several advantages: A structured formal specification is for example much easier to understand. Proving safety properties for a structured system becomes modular as properties of individual components can be used to prove the overall safety property. Also a specification component can be refined independently, keeping the effort of implementing a large system linear to the number of its components.

## 6.2 Specification Components

An example of how to specify a single component can be seen in Figure 4, where parts of the specification of the Field Interface are displayed. Typically a temporal specification component uses algebraic specifications as data types (slot USING). State variables are defined in slot DATA, where variables of type IN, OUT, and INTERNAL are distinguished. Additional state variables can be included by referring to other temporal components (slot INCLUDE). The transitions of a

```

TLSPEC FLD_Interface
PURPOSE "Specification of the Field Interface"
USING natural;
    General_Status;
    Field_State;
    ...
INCLUDE FLD_Time = Time;
        FLD_out = Fld_Sensor_Out
DATA IN    ...
    OUT    Gen_Status      : gen_state_t
    INTERNAL last_target_time : nat;
        FI_State          : FI_State_t;
    ...
ACTIONS
    ...
    TARGET_1m ::=
        /* Precondition */
        last_target_time + 40 < now
    AND FI_State = moving
        /* Postcondition */
    AND FI_State' = emergency
    AND Gen_Status'[Time_out, emergency] = True
    ...
SPEC INITIAL    last_target_time = now
                AND FI_State = idle
                AND ...
    TRANSITIONS [..., TARGET_1m, ...]
                {last_target_time, FI_State, ...}
HIDE last_target_time, FI_State, ...
TLSPECEND

```

Fig. 4. Example of how to specify a single component

temporal component are defined as actions in slot ACTIONS. Arbitrary formulas in first order predicate logic can be given to relate values of variables in one state to values in the next state. Variables are primed, if their values in the next state are referred. Slot SPEC defines the overall behaviour of a component. The initial values of all state variables are defined and a list of actions which are possible is given. A so called stuttering index lists all state variables which are unchanged, if environmental steps are taken. As an alternative to a canonic TLA formula, slot SPEC can also contain arbitrary formulas in linear temporal logic. State variables which are only used for internal purposes can be hidden by listing them in slot HIDE.

```

TLSPEC NAC
  USING ...
  DATA ...
    INTERNAL Common_Queue : prio_queue_t;
      Common_Status : gen_state_t
  COMBINE INT SHARED [INT.Fld_Queue <- Common_Queue,
    INT.Fld_Status <- Common_Status];
    FLD SHARED [FLD.Int_Queue <- Common_Queue
    FLD.Gen_Status <- Common_Status];
  CPI ...
TLSPECEND

```

Fig. 5. Example of how to combine specifications

In a similar manner the other processes Interpolator and Control Panel Interface can be described. How the processes are combined is shown in Figure 5, where the formal specification of the top level processor *NAC* is partly shown. Slot COMBINE refers to the specifications *INT*, *FLD*, and *CPI*. Shared variables are used for communication, state variables from different components are therefore mapped onto one and the same shared state variable of processor *NAC*.

### 6.3 Data Types

Temporal specifications in VSE rely on data types which are defined as algebraic specifications [13] [7]. In Figure 6 the specification of messages to the Interpolator is shown as an example. Specification *INT\_Msg\_Basic* uses a special syntax to define a freely generated type with

- constructor functions *CPI\_ENABLE\_AXIS*, etc.,
- selector functions *ax*, etc., and
- test predicates *CPI\_ENABLE\_AXISP*, etc.

Since freely generated types often occur in formal specifications, they are specially supported. Furthermore algebraic specifications with arbitrary axioms in full first order logic are possible in VSE. Thus it is possible to also define for example sets and real numbers, which occur in the formal model of RCS.

```

BASIC INT_Msg_Basic
PURPOSE "Input Messages for the Interpolator"
USING natural;
    INT_Msg_Args
INT_Msg_t = CPI_ENABLE_AXIS(ax : nat) WITH CPI_ENABLE_AXISP
           | CPI_DISABLE_AXIS(ax : nat) WITH CPI_DISABLE_AXISP
           | CPI_HALT
           | ...
BASICEND

```

Fig. 6. An algebraic specification

#### 6.4 Real Time

In the specification of different components of the Robertino Control System time is needed since there are several time critical behaviors. We have specified a global time instead of a local time to avoid synchronization between components. The global time is realized in a temporal component *Time* which describes the behavior of time represented by a variable *now*. The specification of time assures that time is

1. non-decreasing:  $\Box(now \leq now')$  and
2. non-Zeno:  $\forall i : \Diamond(now > i)$ , where  $i$  and  $now$  are reals.

How the global time is realized is schematically shown in Figure 7. Every

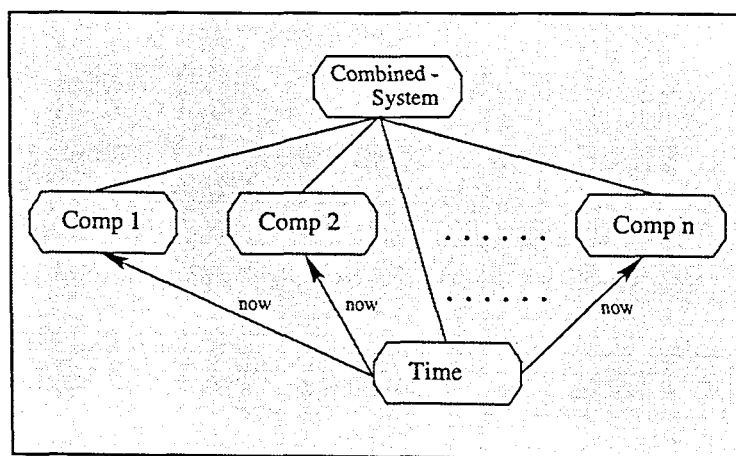


Fig. 7. Inserting time in a system specification

component of the system has an input variable which receives the global time

from the *Time* component. Time bounds are specified in a direct manner within the actions of the components. As an example of such a time condition we have a look at the following action definition taken from the specification of the Field Interface:

$$\begin{aligned} \text{Target}_{1-} \hat{=} & \wedge \text{last\_target\_time} + 40\text{ms} < \text{now} \\ & \wedge \text{FI\_State} = \text{moving} \\ & \wedge \text{FI\_State}' = \text{emergency} \\ & \wedge \text{Gen\_Status}'[\text{Time\_out}, \text{emergency}] = \text{True} \end{aligned}$$

The left hand side of the implication is true if the internal state of the Field Interface is *moving* and if the time that has elapsed since the last target was sent is not more than 40 ms. This is one way to specify timing requirements of the actions constituting the behavior of the system.

## 7 Results and Outlook

In this case study we have constructed a structured distributed formal specification of the central processes of Robertino Control System using VSE. Our effort resulted in more than 2000 lines of specification. The case study has proven, that – due to practical structuring operators in VSE – specifications of that scale can be handled without problems. In addition, a number of errors were found in the existing informal documentation of the manufacturer. Some of the descriptions in the informal documentation are also likely to be misinterpreted. This is not the case in a formal specification.

Our main intention in formally specifying a large real life application was to accompanny and to guide the development of VSE II. As shared variables are extensively used for communication in Robertino, special support was added. The design of the structuring operators for temporal specifications, which we have described in this paper, was also guided by this case study. The concept of the structuring operators is such that compositional assumption guarantee proofs are possible. Thus we expect that formally proving properties for specifications of this size is also possible, but proving the safety model for Robertino in VSE is still to be done. Formally proving the safety model in a next step will help us to further improve deduction in temporal logic in VSE.

## References

1. Martín Abadi and Leslie Lamport. The Existence of Refinement Mappings. *Theoretical Computer Science*, 82(2):253–284, May 1991.
2. Martín Abadi and Leslie Lamport. Conjoining specifications. *ACM Transactions on Programming Languages and Systems*, 17(3):507–534, May 1995.
3. Dieter Hutter, Bruno Langenstein, Claus Sengler, Jörg H. Siekmann, Werner Stephan, and Andreas Wolpers. Deduction in the Verification Support Environment (VSE). In Marie-Claude Gaudel and James Woodcock, editors, *Proceedings Formal Methods Europe 1996: Industrial Benefits and Advances in Formal Methods*. SPRINGER, 1996.

4. Dieter Hutter, Bruno Langenstein, Claus Sengler, Jörg H. Siekmann, Werner Stephan, and Andreas Wolpers. Verification support environment (vse). *High Integrity Systems*, 1(6):523–530, 1996.
5. Dieter Hutter, Heiko Mantel, Georg Rock, Werner Stephan, Andreas Wolpers, Michael Balsler, Wolfgang Reif, Gerhard Schellhorn, and Kurt Stenzel. VSE: Controlling the Complexity in Formal Software Development. In D. Hutter, W. Stephan, P. Traverso, and M. Ullmann, editors, *Proceedings Current Trends in Applied Formal Methods, FM-Trends 98*, Boppard, Germany, 1999. Springer-Verlag, LNCS 1641.
6. Leslie Lamport. The temporal logic of actions. *ACM Transactions on Programming Languages and Systems*, 16(3), 1994.
7. P. D. Mosses. CoFI : The common framework initiative for algebraic specification. In H. Ehrig, F. v. Henke, J. Meseguer, and M. Wirsing, editors, *Specification and Semantics*. Dagstuhl-Seminar-Report 151, 1996. <http://www.brics.dk/Projects/CoFI>.
8. W. Reif, G. Schellhorn, K. Stenzel, and M. Balsler. Structured specifications and interactive proofs with KIV. In W. Bibel and P. Schmitt, editors, *Automated Deduction—A Basis for Applications*. Kluwer Academic Publishers, 1998.
9. Georg Rock, Werner Stephan, and Andreas Wolpers. Tool support for the compositional development of distributed systems. In *Tagungsband 7. GI/ITG-Fachgespräch Formale Beschreibungstechniken für verteilte Systeme*, number 315 in GMD Studien. GMD, 1997.
10. Georg Rock, Werner Stephan, and Andreas Wolpers. Modular Reasoning about Structured TLA Specifications. In R. Berghammer and Y. Lakhnech, editors, *Tool Support for System Specification, Development and Verification*, Advances in Computing Science, pages 217–229. Springer, WienNewYork, 1998.
11. Emilio Ruiz-Morales. A Software Development Approach for Robotics Control Systems. In *Proceedings of Safecom*, pages 317–330. Springer, 1995.
12. S. Scheer and S. Bonino. Formal Software Development and Correctness Verification Using the VSE Methods and Tool. In *Technical Note No. I.97.95*,. European Commission, Joint Research Centre, Ispra, 1997.
13. M. Wirsing. *Algebraic Specification*, volume B of *Handbook of Theoretical Computer Science*, chapter 13, pages 675 – 788. Elsevier, 1990.