Appeared in the Proceedings of the 5th Intl Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems FTRTFT'98, LNCS 1486, p. 123--132, Lyngby, Denmark, Sept, 1998

Designing a Provably Correct Robot Control System using a 'Lean' Formal Method*

Antonio Cau, Chris Czarnecki and Hussein Zedan

Software Technology Research Laboratory,
SERCentre, De Montfort University,
The Gateway, Leicester LE1 9BH, UK

Abstract. A development method for the construction of *provably correct* robot control systems together with its supporting tool environment are described. The method consists of four stages: 1. specification, 2. refinement, 3. simulation and 4. code. The method is centered around the notion of *wide-spectrum* formalism within which an abstract Interval Temporal Logic (ITL) representation is intermixed freely with the concrete Temporal Agent Model (TAM) representation of the system under consideration. The method with its associated tool support is applied to the design of a robot control system.

1 Introduction

Designing software to control *real-time, reactive embedded* applications is non-trivial. And as the complexity of such systems increases, the present industrial practice for their development gives cause for concern, especially, if they are to be used in safety-critical applications. In order for the design of these systems to be optimized, it is necessary to take into account the interdependence of the hardware and software. Thus, the system needs to be assessed at all stages of the development life-cycle in order to minimize the potential for errors. This has resulted in the development of a wide range of techniques which aim to support the analysis and design of both systems and their associated software. These vary from those with sound mathematical basis (*formal methods*) to *structured methodologies*. The later, while useful, do not provide a satisfactory and comprehensive solution. The former, on the other hand, are recognized as the most likely solution to the problem, but insufficient expertise and a lack of tool support have limited their deployment, specially in a highly specialized applications. For example, in the Automotive Industry, the MISRA *"Development Guidelines for Vehicle Based Software"* recommend the use of formal methods in the specification and analysis of systems at safety integrity levels 3 (difficult to control) and level 4 (uncontrollable). At present this represents an extremely

* Funded by EPSRC Research Grant GR/K25922: A Compositional Approach to the Specification of Systems using ITL and Tempura.
E-mail: {cau, cc, zedan}@dmu.ac.uk

difficult and costly step which has not yet been tried by system developers. These weaknesses in current analysis techniques therefore represent a significant threat for the deployment of advanced automotive electronics in the future.

The general objective should be therefore to bring real-time systems and software engineering activities to a similar level of maturity to that of traditional engineering disciplines. The key to this is predictability through the development of, what might be called, *'lean'* formal method which will be strongly supported by a suite of powerful and tightly integrated, practicable and affordable software tools.

In the present paper we outline our attempt to develop such *'lean'* formal method. The proposed formal design framework has an underlying abstract computational model leads to designs which can be analyzed for their schedulability. We will illustrate our technique by designing a *provably correct robot control system*.

The technique provides an integration between a logic-based formalism, namely, Interval Temporal Logic (*ITL*), Tempura (an executable subset of ITL) and a real-time refinement calculus, known as *TAM*. The technique is supported by various tools which forms the ITL-workbench.

1.1 Our Approach

The technique is depicted in Figure 1, and can be summarized as follows:

1. Specification is expressed as an ITL formula.
2. Refine specification into TAM code if possible otherwise revise the specification in 1.
3. Simulate the TAM code. If satisfactory then proceed otherwise either refine it in 2 to other TAM code or revise it in 1.
4. Translate TAM code into real programming language like C, ADA, etc.

We should note here that the TAM code represent the last stage in our formal development method. The TAM code could either be executed directly or be translated into an industrially accepted target programming language such as C, Ada, etc. Care must be taken however that the semantics of the target language is equivalent to TAM semantics.

Various observations are in order:

1. Once we completed the formal specification phase, various properties could be proven about the specification itself. This can provide an extra assurance that the final specification meets the required informal requirements. This is achieved within out ITL-workbench using PVS.
2. The ITL specification could directly be executed using Tempura. This gives an early confidence on the validity of the specification.
3. At each refinement step, we can simulate the resulting (sub)system. This gives some guidelines on the choice of the subsequent refinement rules.

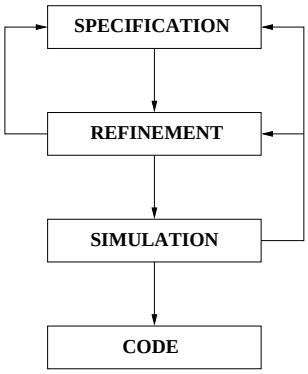

Fig. 1. The design strategy

4. The refinement calculus TAM used in this technique is based on *scheduling-oriented* model which was introduced in Lowe and Zedan [2]. It also uses *delayed specification* technique which we believe it makes formalizing many requirements much easier and less error prone.

1.2 Paper Structure

The format of the paper is as follows. In Section 2, we introduce the refinement calculus TAM together with its ITL semantics domain and some useful algebraic equations. Section 3 gives an informal description of the robot. The formal development of the robot control system will be given in Section 4. This section highlights the power of the simulation tool Tempura which supports the adopted design method.

2 Interval Temporal Logic and Temporal Agent Model

The formal development framework (TAM) proposed here provides the developer with:

1. a mathematical logic with which a formal description of the intended behaviors of the system under consideration could be precisely and unambiguously expressed. This is considered to be the highest level of abstraction of system development. For this purpose we have chosen a temporal logic known as Interval Temporal Logic (ITL) [3,4] for its simplicity and the availability of an executable subset which assists in the simulation phase.
2. a wide-spectrum language (WSL) with formally defined syntax and semantics. The WSL allows the developer to intermix between abstract (i.e. logical) and concrete constructs giving greater flexibility in compositional design.
3. sound refinement rules which allows stepwise development of systems.

2.1 Interval temporal logic

This section introduces the syntax and informal semantics of Interval temporal logic (ITL). Our selection of ITL is based on a number of points. It is a flexible notation for both propositional and first-order reasoning about periods of time found in descriptions of hardware and software systems. Unlike most temporal logics, ITL can handle both sequential and parallel composition and offers powerful and extensible specification and proof techniques for reasoning about properties involving safety, liveness and projected time [5]. Timing constraints are expressible and furthermore most imperative programming constructs can be viewed as formulas in a slightly modified version of ITL [1]. Tempura provides an executable framework for developing and experimenting with suitable ITL specifications. In addition, ITL and its mature executable subset Tempura [4] have been extensively used to specify the properties of real-time systems where the primitive circuits can directly be represented by a set of simple temporal formulae.

$$\begin{array}{rl} \textit{Expressions} & e ::= \mu \mid a \mid A \mid g(e_1, \ldots, e_n) \mid \imath a{:}\, f \\ \textit{Formulae} & f ::= p(e_1, \ldots, e_n) \mid \neg f \mid f_1 \wedge f_2 \mid \forall v \bullet f \mid \mathsf{skip} \mid f_1 \,;\, f_2 \mid f^* \end{array}$$

Fig. 2. Syntax of ITL

An interval is considered to be a (in)finite sequence of states, where a state is a mapping from variables to their values. The length of an interval is equal to one less than the number of states in the interval (i.e., a one state interval has length 0).

The syntax of ITL is defined in Fig. 2 where μ is an integer value, a is a static variable (doesn't change within an interval), A is a state variable (can change within an interval), v a static or state variable, g is a function symbol and p is a predicate symbol.

The informal semantics of the most interesting constructs are as follows:

- $\imath a{:}\, f$: the value of a such that f holds.
- skip: unit interval (length 1).
- $f_1\,;f_2$: holds if the interval can be decomposed ("chopped") into a prefix and suffix interval, such that f_1 holds over the prefix and f_2 over the suffix, or if the interval is infinite and f_1 holds for that interval.
- f^*: holds if the interval is decomposable into a finite number of intervals such that for each of them f holds, or the interval is infinite and can be decomposed into an infinite number of finite intervals for which f holds.

These constructs enable us to define programming constructs like assignment, if then else, while loops etc.

$$\mathcal{A} ::= w : f \mid \mathsf{skip} \mid x := e \mid x \Leftarrow s \mid e \Rightarrow s \mid \mathcal{A} \,;\, \mathcal{A}' \mid \mathsf{var}\, x \,\mathsf{in}\, \mathcal{A} \mid$$
$$\mathsf{shunt}\ s \,\mathsf{in}\, \mathcal{A} \mid [t]\ \mathcal{A} \mid \mathsf{if_t} \ \Box_{i \in I}\, g_i \,\mathsf{then}\, \mathcal{A}_i \ \mathsf{fi} \mid \mathcal{A} \sqcap \mathcal{A}' \mid \mathcal{A} \rhd^s_t \mathcal{A}' \mid$$
$$\mathcal{A} \parallel \mathcal{A}' \mid \Delta t \mid \ \mathsf{loop\,for}\, n \,\mathsf{period}\, t \ \ \mathcal{A}$$

Fig. 3. Syntax of TAM

2.2 Temporal Agent Model

The temporal agent model (TAM) is a well established formalism [8,9,2] for the development real-time safety-critical systems.

At any instant in time a system can be thought of as having a unique *state*. The system state is defined by the values in the shunts (time-stamped variables) and the state variables of the system, the so called *frame*. This frame defines the variables that can possibly change during system execution, the variables outside this frame will certainly not change. *Computation* is defined to be a sequence of system states , i.e., an *interval* of states. ITL enables us to describe this set of computations in an eloquent way.

The syntax of TAM is defined in figure 3 where w is a set of computation variables and shunts; f is an ITL formula; t is a time; x is a variable; e is an expression on variables; s is a shunt; I is some finite indexing set; g_i is an ITL formula without temporal operators; and n is a natural number.

The informal semantics of some of the TAM constructs is as follows:

- $w : f$ is a specification statement. It specifies that only the variables in the *frame* w may be changed, and the execution must satisfy f.
- The agent $x \Leftarrow s$ performs an input from shunt s, storing the value in x; the type of x must be a stamp–value pair.
- The agent $e \Rightarrow s$ writes the current value of expression e to shunt s, increasing the stamp by one.
- The agent $[t]\ \mathcal{A}$ gives agent $\mathcal{A}$ a duration of t: if the agent terminates before t seconds have elapsed, then the agent should idle to fill this interval; if the agent does not terminate within t seconds, then it is considered to have failed.

TAM refinement and algebraic rules In this section we explore some of the algebraic properties of the TAM agents. But first, we define the *refinement* ordering relation $\sqsubseteq$ as logical implication:

$$f_0 \sqsubseteq f_1 \mathrel{\widehat{=}} f_1 \ \supset \ f_0$$

The following are some basic refinement rules:

$$
\begin{array}{lll}
(\sqsubseteq -1) \vdash (f_0 \sqsubseteq f_1) \text{ and } \vdash (f_1 \sqsubseteq f_2) & \Rightarrow & \vdash (f_0 \sqsubseteq f_2) \\
(\sqsubseteq -2) \vdash (f_0 \sqsubseteq f_1) \text{ and } \vdash (f_2 \sqsubseteq f_3) & \Rightarrow & \vdash (f_0 \wedge f_2) \sqsubseteq (f_1 \wedge f_3) \\
(\sqsubseteq -3) \vdash (f_0 \sqsubseteq f_1) \text{ and } \vdash (f_2 \sqsubseteq f_3) & \Rightarrow & \vdash (f_0 \vee f_2) \sqsubseteq (f_1 \vee f_3) \\
(\sqsubseteq -4) \vdash f_1 \sqsubseteq f_2 & \Rightarrow & \vdash f_0 \,;\, f_1 \sqsubseteq f_0 \,;\, f_2 \\
(\sqsubseteq -5) \vdash f_1 \sqsubseteq f_2 & \Rightarrow & \vdash f_1 \,;\, f_0 \sqsubseteq f_2 \,;\, f_0 \\
(\sqsubseteq -6) \vdash f_0 \sqsubseteq f_1 & \Rightarrow & \vdash f_0^* \sqsubseteq f_1^* \\
(\sqsubseteq -7) \vdash f_0 \sqsubseteq f_1 & \Rightarrow & \vdash \forall v \bullet f_0 \sqsubseteq \forall v \bullet f_1
\end{array}
$$

Assignment: The assignment is introduced with the following law

$$(:= -1)\ x := exp \quad \equiv \quad \bigcirc x = exp$$

If then–conditional: The conditional is introduced with the following law

$$
\begin{array}{ll}
(\mathsf{if}\ -1)\ \mathsf{if}\ \ f_0\ \mathsf{then}\ f_1 \,\square\, f_2\ \mathsf{then}\ f_3\ \ \mathsf{fi} & \equiv (f_0 \wedge f_1) \vee (f_2 \wedge f_3) \\
(\mathsf{if}\ -2)\ \mathsf{if}\ \ f_0\ \mathsf{then}\ f_1\ \mathsf{else}\ f_2\ \ \mathsf{fi} & \equiv (f_0 \wedge f_1) \vee (\neg f_0 \wedge f_2) \\
 & \equiv \mathsf{if}\ \ f_0\ \mathsf{then}\ f_1 \,\square\, \neg f_0\ \mathsf{then}\ f_2\ \ \mathsf{fi}
\end{array}
$$

While loop: The following law introduces the while loop

$$
\begin{array}{lll}
(\mathsf{while}\ -1)\ \mathsf{while}\ \ f_0\ \mathsf{do}\ f_1 & \equiv & (f_0 \wedge f_1)^* \wedge \mathit{fin}\, \neg f_0 \\
(\mathsf{while}\ -2)\ \mathsf{while}\ \ \mathit{true}\ \mathsf{do}\ f_1 & \equiv & f_1^*
\end{array}
$$

Parallel: The following are some laws for the parallel agent.

$$
\begin{array}{lll}
(\parallel -1)\ f_0 \parallel f_1 & \equiv & f_0 \wedge f_1 \\
(\parallel -2)\ f \parallel \mathit{true} & \equiv & f \\
(\parallel -3)\ (f_0 \parallel f_1) \parallel f_2 & \equiv & f_0 \parallel (f_1 \parallel f_2)
\end{array}
$$

Variable introduction: The following is the local variable introduction law.

$$(\mathsf{var}\ -1)\ \mathsf{var}\ x\ \mathsf{in}\ \mathcal{A} \quad \equiv \quad \exists x \bullet \mathcal{A}$$

The next section introduces the robot system which will be used as a case study.

3 Robot description

The tele-operated robot is a tracked device which was originally developed for military use. The carriage can easily traverse over rough terrain. The vehicle schematic is shown in Fig. 4. The vehicle has on-board a manipulator arm that has three degrees of freedom controlled by hydraulic actuators. The electric drive motors, manipulator actuators and on-board lamps are controlled manually by the operator via a control box that is linked to the vehicle. Currently one controller caters for the main control, one for the infrared sensor interfacing and processing, and a third for the on-board camera control.

The actual vehicle is driven by two motors, left and right, indicated as L and R in Fig. 4. Both of these motors can move forwards and in reverse. The vehicle is steered by moving one motor faster than the other.

From a control point of view, commands are issued to the motors via a operator joystick (L and R of the operator console in Fig. 4) which issues integers values in the range $0 \ldots 127$ for forward motion (127 max. speed) and $0 \ldots -128$ for reverse motion. It is possible to drive only one motor at a time, in such a case the robot will turn. The speed of the motors is directly proportional to the value written to them.

The robot is equiped with 8 infra red sensors. These return an integer value in the range $0 \ldots 255$ depending on whether an obstacle is present or not. 0 indicates no obstacle, 255 indicates obstacle very near. We normally operate with a threshold of around 100, above which we take notice of the sensor readings, i.e., an obstacle is of interest. At this point reactive control takes over from the manual control by moving the vehicle away from the obstacle until the 100 threshold is not set. The sensor positions are as follows: N, NE, E, SE, S, SW, W and NW, covering the body of the robot and shown in Fig. 4.

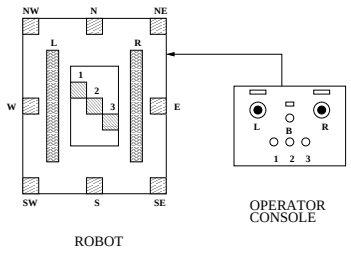

Fig. 4. The robot control system

4 Robot specification and design

This section presents the specification and the design (refinement) of the driving part of the robot control system. The hazardous nature of the task, coupled with the integration of reactive control and human commands presents a challenging safety critical application. We will use our design strategy of Sect. 1.1.

4.1 Specification

The specification of the robot control system consists of 3 parts:

1. Motor control: If the sensor detects an object then the control system takes over control otherwise if the operator requests a new movement then action this.

Let *l-i-c* and *r-i-c* denote respectively the left and right motor commands issued by the infra-red control. Let *i-act* denote the presence/absence of an object. Let *l-o-c* and *r-o-c* denote the left and right motor command issued by the operator and let *o-act* denote an active operator request. Let $move(l, r)$ denote the sending of left l and right r motor commands to the two motors.

The motor control is specified as:

$$\begin{array}{lll} MCS \mathrel{\widehat{=}} (& (\textit{i-act} \wedge move(\textit{l-i-c}, \textit{r-i-c})) \vee \\ & (\textit{o-act} \wedge move(\textit{l-o-c}, \textit{r-o-c})) \vee \\ & (\neg \textit{i-act} \wedge \neg \textit{o-act}) \qquad\qquad)^* \end{array}$$

Note-1: if *i-act* and *o-act* are both enabled we have a non-deterministic choice. In the refinement process below we take the design decision that in this case the operator commands override.

2. Infra-red control: read the sensors and for each sensor that is greater than the threshold of 100 adjust the motor commands accordingly. For example if the north sensor detects an object we should move in the south direction as an avoidance strategy.

Let *ir-c*(i) denote the sensor i (N: i=0, NE: i=1, E: i=2, SE: i=3, S: i=4, SW: i=5, W: i=6, NW: i=7). Let *mvl*(i) and *mvr*(i) denote respectively the left and right steering commands corresponding to sensor i.

The infra-red control is specified as:

$$\begin{array}{lll} ICS \mathrel{\widehat{=}} (& \textit{i-act} = (\bigvee_i \textit{ir-c}(i) > 100) \wedge \\ & \textit{l-i-c} = (\sum_i : \textit{ir-c}(i) > 100 : \textit{mvl}(i)) \wedge \\ & \textit{r-i-c} = (\sum_i : \textit{ir-c}(i) > 100 : \textit{mvr}(i)) \quad)^* \end{array}$$

3. Operator control: if the operator requests some changes then process them.

Let *l-o-c* and *r-o-c* denote respectively the left and right steering commands received from the operator. Let *ll-o-c* and *lr-o-c* denote respectively the last left and last right steering commands received from the operator.

The specification of operator control is as follows:

$$\begin{array}{lll} OCS \mathrel{\widehat{=}} & \exists \textit{ll-o-c}, \textit{lr-o-c} \bullet (& \\ & \quad \textit{ll-o-c} = 0 \wedge \textit{lr-o-c} = 0 \wedge & \\ & \quad (\textit{o-act} = (\textit{l-o-c} \neq \textit{ll-o-c} \vee \textit{r-o-c} \neq \textit{lr-o-c}) \wedge & \\ & \quad\ \bigcirc \textit{ll-o-c} = \textit{l-o-c} \wedge \bigcirc \textit{lr-o-c} = \textit{r-o-c} \qquad)^* & \\ &) & \end{array}$$

The overall specification is $\textit{Robotcs} \mathrel{\widehat{=}} MCS \wedge ICS \wedge OCS$.

4.2 Refinement

In this section we show the refinement process. We will show only the refinement of MCS due to lack of space.

First we refine * into a while loop using law (while -2) and strengthened the guard of the infra-red move (only taken if the operator is also inactive):

$$
\begin{array}{l}
MCS \sqsubseteq \\
\quad \textsf{while } \mathit{true} \textsf{ do } (\ (\mathit{i\text{-}act} \wedge \neg \mathit{o\text{-}act} \wedge \mathit{move}(\mathit{l\text{-}i\text{-}c}, \mathit{r\text{-}i\text{-}c})) \vee \\
\qquad\qquad\qquad\quad (\mathit{o\text{-}act} \wedge \mathit{move}(\mathit{l\text{-}o\text{-}c}, \mathit{r\text{-}o\text{-}c})) \vee \\
\qquad\qquad\qquad\quad (\neg \mathit{i\text{-}act} \wedge \neg \mathit{o\text{-}act}) \\
\quad)
\end{array}
$$

Then we introduce the if then with the (if -1) law.

$$
\begin{array}{l}
\sqsubseteq \\
\textsf{while } \mathit{true} \textsf{ do } (\ \textsf{if } \mathit{i\text{-}act} \wedge \neg \mathit{o\text{-}act} \quad \textsf{then } \mathit{move}(\mathit{l\text{-}i\text{-}c}, \mathit{r\text{-}i\text{-}c}) \\
\qquad\qquad\qquad\quad \Box\ \mathit{o\text{-}act} \qquad\qquad\quad \textsf{then } \mathit{move}(\mathit{l\text{-}o\text{-}c}, \mathit{r\text{-}o\text{-}c}) \\
\qquad\qquad\qquad\quad \Box\ \neg \mathit{i\text{-}act} \wedge \neg \mathit{o\text{-}act} \textsf{ then } \mathit{true} \\
\qquad\qquad\qquad\quad \textsf{fi} \\
)
\end{array}
$$

The last refinement step consists of choosing a specific execution time for the body of the while loop. An execution time of t_{m} results in the following code.

$$
\begin{array}{l}
\sqsubseteq \\
\textsf{while } \mathit{true} \textsf{ do } (\ \textsf{if}_{t_{\mathrm{m}}} \ \mathit{i\text{-}act} \wedge \neg \mathit{o\text{-}act} \quad \textsf{then } \mathit{move}(\mathit{l\text{-}i\text{-}c}, \mathit{r\text{-}i\text{-}c}) \\
\qquad\qquad\qquad\quad \Box\ \ \mathit{o\text{-}act} \qquad\qquad\quad \textsf{then } \mathit{move}(\mathit{l\text{-}o\text{-}c}, \mathit{r\text{-}o\text{-}c}) \\
\qquad\qquad\qquad\quad \Box\ \ \neg \mathit{i\text{-}act} \wedge \neg \mathit{o\text{-}act} \textsf{ then } \mathit{true} \\
\qquad\qquad\qquad\quad \textsf{fi} \\
)
\end{array}
$$

ICS and OCS can be refined likewise. The results of each refinement are then composed together with rules ($\sqsubseteq -2$) and ($\| -1$) resulting in the concrete TAM code of the robot control system.

4.3 Simulation

We use the Tempura[1] interpreter to simulate the TAM concrete code. This interpreter is written in C by Roger Hale and based on a Lisp version of Ben Moszkowski [4]. A graphical front end written in Tcl/Tk has been developed. Tempura offers a means for rapidly developing and testing suitable ITL/TAM specifications. As with ITL/TAM, Tempura can be extended to contain most imperative programming features and yet retain its distinct temporal feel. The use of ITL/TAM and Tempura combine the benefits of traditional proof methods balanced with the speed and convenience of computer-based testing through execution and simulation. The entire process can remain in one powerful logical and compositional framework. A practitioner can allocate his/her time and other resources to one or the other approach based on the merits of the system under consideration. Output from Tempura can be used as "documentation by example" to supplement other specification techniques.

[1] available from `http://www.cms.dmu.ac.uk/~cau/itlhomepage/index.html`

4.4 Code

In this section we will translate TAM code into C. This translation is currently performed by hand but a tool that automatically converts to C, is under development. The choice of C is governed by the compiler available for the embedded target system. Other target languages such as Ada are equally valid.

5 Conclusion and future work

Designing software to control *real-time, reactive embedded* applications is non-trivial. As the complexity of such systems increases, the present industrial practice for their development gives cause for concern, especially, if they are to be used in safety-critical applications. Part of the reason for this can be considered due to the lack of appropriate *assurance* techniques. It is believed that the next generation of systems will indeed test our capabilities to the limit.

In this paper we have introduced a development method for the construction of *provably correct* robot control systems together with its supporting tool environment. The method is centered around the notion of *wide-spectrum* formalism within which an abstract (logical) representation is intermixed freely with the concrete representation of the system under consideration. The transformation from the abstract to the concrete representations is achieved by applying a series of correctness preserving refinement laws. In this way the resulting concrete representation is guaranteed to satisfy the required behavioral specification expressed at the logical level. At each stage of the refinement, the developer may *simulate* the resulting system. This will provide an extra level of assurance as well as a guideline to which refinement path the developer may wish to apply.

The choice of ITL was due to its simplicity and its capability of specifying various real-time reactive embedded requirements. Being a temporal logic, system's dynamics can be easily expressed and reasoned about. In addition, ITL is supported by an executable programming environment known as Tempura [4] which allows a fast prototyping and simulation of the design.

We believe that our method provides a very useful tool in developing provably correct real-time reactive systems. When combined with the *delayed specification* technique, introduced in [2], makes formalizing many requirements much easier and less error prone.

References

1. Cau, A. and Zedan, H.: *Refining Interval Temporal Logic Specifications.* In proc. of Fourth AMAST Workshop on Real-Time Systems, Concurrent, and Distributed Software (ARTS'97), LNCS **1231**, Mallorca, Spain, May 21–23, (1997) 79–94
2. Lowe, G. and Zedan, H.: *Refinement of Complex Systems: a Case Study.* The Computer Journal, **38**:10, (1995)
3. Moszkowski, B.: *A Temporal Logic for Multilevel Reasoning About Hardware.* IEEE Computer **18**, (1985) 10–19

4. Moszkowski, B.: *Executing Temporal Logic Programs*. Cambridge Univ. Press, UK, (1986)
5. Moszkowski, B.: *Some Very Compositional Temporal Properties*. In Programming Concepts, Methods and Calculi, Ernst-Rüdiger Olderog (ed.), IFIP Transactions, Vol. **A-56**, North-Holland, (1994) 307–326
6. Rushby, J.: *A Tutorial on Specification and Verification using PVS*. In proc. of the FME '93 symposium: Industrial-Strength Formal Methods, J.C.P. Woodcock and P.G. Larsen (eds.), LNCS **670**, Odense, Denmark, (1993) 357–406. Check homepage: `http://www.csl.sri.com/pvs.html`
7. Sheridan, T.B.: *Telerobotics, Automation, and Human Supervisory Control*. The MIT Press, Cambridge, Massachusetts, (1992)
8. Scholefield, D.J., Zedan, H. and He, J.: *Real-time Refinement: Semantics and Application*. LNCS **711**, (1993) 693–702
9. Scholefield, D.J., Zedan, H. and He, J.: *A Specification Oriented Semantics for the Refinement of Real-Time Systems*. Theoretical Computer Science **130**, (1994)