FORMAL METHODS IN SYSTEM DESIGN: A CASE STUDY

VINCENZA CARCHIOLO, MICHELE MALGERI AND GIUSEPPE MANGIONI
Istituto di Informatica e Telecomunicazioni Facoltà di Ingegneria
Università di Catania
Viale Andrea Doria, 6 - I95125 Catania - Italy

Abstract: - The development of tools for the design of both hardware and software systems draws great benefits from the use of formal methods, especially if they offer a descriptive capacity which covers real applications. On the basis of the T-LOTOS language, a language called TTL has been developed, which adds new constructs and tools to the considerable existing expressiveness of the original language, thus making it suitable for the specification of hardware and software systems of real complexity. Some of the extension proposed covers the aspects of modularization of the specification, the introduction of iterative construct and a first moving to object paradigma. An extensive example of the use of TTL is presented to show its characteristics.

Key-Words: - Modeling, Formal Verification, Verification and Validation, Computer Algebra, Analysis and design tools.

1. Introduction

The problem of designing and specifying complex systems for which constraints of time and synchronization - with the outside world or other systems - have to be respected, is a highly debated topic. It is generally dealt with by imposing precise design methods which make it possible to test that the results obtained meet the requirements laid down when the system is defined (CAD methods and software engineering). Fundamental in any design method is the choice of a specification technique which allows systems to be described unambiguously and clearly. One sector where the problem has been dealt with is that of communication protocols, where the first studies on the application of formal description techniques to design were made, and in this context a number of particularly suitable languages (or FTDs - Formal Description Techniques) have been standardized, such as LOTOS [1][2] and ESTELLE [3].

Experience acquired with communication protocols suggested extending the field of application of FDTs to other sectors such as hardware design [4].

Until a few years ago hardware systems were designed using CAD tools, mainly based on graphic editors, which enabled designers to define the electrical scheme of the connections between the various components, and programs whereby it was possible to extract a certain amount of information from them. This information was used to test and simulate the behaviour of the device. Interpretation of the results depended on the ability of designers who used their experience to guarantee the correctness of the design. If the system needed interaction between the hardware and software components, the two parts were usually separated at the very beginning of the design cycle. They were therefore developed independently, with little interaction up to the integration phase. This approach limited the opportunity to explore possible hw/sw trade-offs, such as moving certain functions from the software to the hardware domain (and vice versa), and greatly complicated both design and assessment of the cost due to interfacing between the two parts. The development of circuit technology led to an increase in the number of elementary devices on a single chip and thus to a need to design a device not at the gate level but as a set of appropriately connected blocks (memories, adders, etc). This led to the definition of languages to describe hardware systems (e.g. VHDL [5] and Verilog [6]). The increasing complexity of systems has pointed to the necessity of applying to the design of hardware devices the knowledge acquired in the development of distributed systems, such as the use of formal tools.

Recently considerable interest has been shown in the proposal to integrate the design of software and hardware components in such a way that it is not necessary to make a preliminary choice, but rather to leave the decision to implement certain components in hardware or software to the final stages of design. One of the most widely investigated fields in this context is that of control-dominated embedded systems , i.e. systems which react to external stimuli by performing a certain action, choice of which does not involve very complex calculations.

Our work fits into this context: its aim is to propose a framework which will be of help in the development of systems belonging to the category mentioned above. In this paper we describe the language we have chosen as specification language (called Templated T-LOTOS) and its application to an example of an home automation system (integrated management of the heating, lighting and alarm etc. systems). In this example we show the peculiar features of the language and the advantages it offers, mainly in making the description modular and modifiable. We also discuss the various stages which led to the development of the example and how it can easily be extended, thanks to the characteristics of the language.

2. Templated T-LOTOS

The first problem that had to be tackled was that of choosing a specification language which must be suitable for both software and hardware. Several FDTs suitable for this purpose have been proposed in literature, one of the most interesting being XCIRCAL [7] based on the process algebra called CIRCAL [8]. With the formal model CIRCAL it is not possible to deal with quantitative time references. Another language that has been used to specify hw/sw systems is ESTEREL [9], which has been proved to be equivalent to the Extended FSM model.

The language we have chosen as a basis for the design of hardware-software systems (see [10][11]), in the attempt to overcome the limits of the languages mentioned above, is an extension of T-LOTOS [12] called TTL[13] (Templated T-LOTOS). T-LOTOS derives from LOTOS which allows time constraints to be specified. It has already been widely used to specify complex systems, mainly in the area of communication protocols [14] [15], but there are also examples of application to hardware systems [4].

When applied to the description of mixed systems of real size, T-LOTOS has certain limits relating to the impossibility of making modular specifications. This involves economic problems as libraries of already developed parts cannot be re-used. In addition, the lack of tools to modularize specifications makes it difficult to share the specification work out between people, as it requires greater interaction between designers. This aspect is fundamental because the complexity of the systems to which we are applying these techniques is such as to require the collaboration of a large number of people, often specializing in different areas. These considerations led to the proposal to extend T-LOTOS. The features of TTL which made it useful for the application in the scenario above presented are:

1. *Formal base.* This is the key feature of the language; it allows us to describe a system in a clear and unambiguous way. Many systems are often used in life critical situations, where reliability and safety are the most important criteria. Formally defined languages allows us to meet this requirement.
2. *High degree of abstraction.* This feature allows us to reduce the complexity of the description and permits us both to describe software and hardware systems due to independence from target architecture.
3. *Concurrency.* This features permits us to model real life systems which are generally made by several parts evolving concurrently.
4. *Time attributes.* This features allows us to describe time requirements of real life systems.
5. *Modularity.* This feature permits to use already specified and tested parts so allowing to save time.

Some of the previously mentioned features are already owned by LOTOS and T-LOTOS.

The TTL extensions allow a modular description and also offer the possibility of describing generic modules (template). To add further flexibility to the language, extensions to the basic constructs are also proposed to make the specification of frequent situations, such as repetitive cycles, more compact. The ways in which these extensions are integrated in the T-LOTOS model are discussed in [13]. One particular feature of the extensions made is that they do not preclude use of the tools available for T-LOTOS, for instance LOLA[16] which allows a complete analysis and simulation of the specification.

3. A Case Study

We will now give an example of the use of TTL in describing a complex system. It is the description of a home automation system (integrated management of the heating, lighting and alarm etc. systems) which manages a large number of sensors and actuators with great flexibility.

The main components of the system are the master, the peripheral control panels and the controllers of the sensors and actuators. Each sensor and actuator in the system is identified by a 16-bit address, the first 8 of which identify one of the 256 possible peripheral panels, and the last 8 one of the 256 possible actuators or sensors connected with it. It is therefore possible to have 256 control panels, each of which can control up to 256 sensors and actuators. The addresses of both are assigned in the instantiation phase.

The following is a detailed list of the components and the relative TTL specification; for reasons of

space and simplicity the part concerning the definition of data types has been omitted.

3.1 Master

The master is the component which coordinates all the peripheral control panels to which it is connected. Its tasks are:

- to read from a serial line (Sl) the values with which to configure the various sensor controllers;
- to distribute the sensor configurations to the peripheral control panels (Mout);
- to dialogue with the various control panels through a dedicated line for each panel (Min_k).

Part (a) of fig. 1 is a graphic representation of the master, while part (b) shows the declaration and TTL definition of the *Master* module.

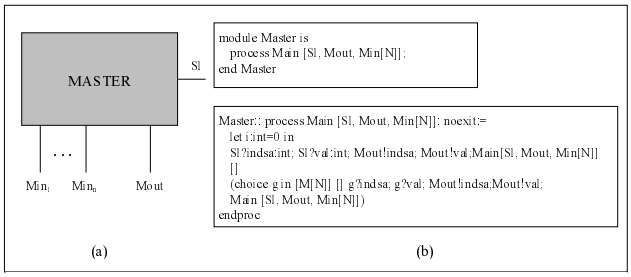

Fig.1: Master Module Schema, its declaration and definition

The Master process is a TTL template; as it can be parametrized with respect to the number of gates, it is possible to describe the set of Masters for varying numbers of control panels as a single process.

It should be noted that in the template context M[N] is a list of N gates which will be defined when the templates are instanced; so M[N] can be used anywhere a list of gates is provided for. In fig. 1, for example, **choice** g **in** M [N] will be translated into **choice** g **in** $[g_1,g_2,...,g_n]$ where g_1, g_2,...,g_n are the elements of the vector M.

3.2 Peripheral Control Panel

A peripheral control panel is the part of the system which controls the sensors and actuators. Each one has an 8-bit address by which it is selected by the master. The specification of the control panel is given in such a way as to allow flexible use in various situations. Part (a) of fig. 2 gives a scheme of the panel, while part (b) shows the declaration of the *Cent_Per* module. The parameter ind is the address of the panel, given when it is instanced.

The control panel has three parts:

- a Sensor Manager which deals with the sensors;
- a Act Manager which deals with the actuators;
- a Routing Table which deals with the correspondence between sensors and actuators.

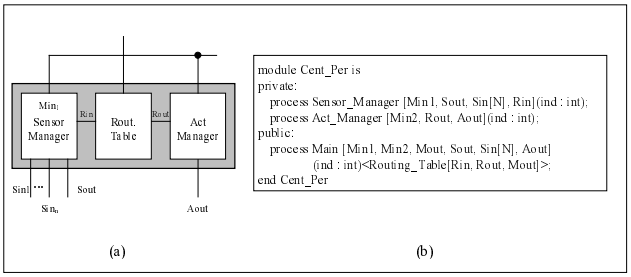

Fig.2: Control Panel schema and its declaration

A definition of the Main process is given in fig. 3 whereas fig. 4 gives the definition of the other processes declared in fig. 3.

```
Cent_Per:: process Main [Min1, Min2, Mout, Sout, Sin[N], Aout](ind : int)
                    <Routing_Table [Rin, Rout, Mout]>: noexit:=
    (Sensor_Manager [Min1, Sout, Sin[N], Rin](ind)
    |[Rin]|Routing_Table [Rin, Rout, Mout]
    |[Rout]|Act_Manager[Min2, Rout, Aout](ind))
endproc
```

Fig.3: Control Panel main process definition

It should be noted that in this process another characteristic of templates is used, that is the chance to parametrize with respect to the process names. In this case the actuator-sensor connections are only defined when the control panel is instanced, thus allowing a highly generic description of the device. In the definition of Cen_Per a loop is used to specify the polling cycle in an extremely simple and natural way.

```
Cent_Per:: process Sensor_Manager [Min1, Sout, Sin[N], Rin](ind : int): noexit:=
  let i:nat=0 in
  loop (i<N; i+1; S_M1[Min1, Sout, Sin[N], Rin](ind,i))
                  >> Sensor_Manager [Min1, Sout, Sin[N], Rin](ind )
  where process S_M1[](ind,i: int):noexit:=
   Sin[i]?indsa:int; Sin[i]?val:int;
    (Rin!(indsa+ind); Rin!val;exit
    []
    Min1?indsa1:int; Min1?val1:int;
     ([((indsa1)>=(ind*256)) AND (indsa1<((ind+1)*256))]->
                  Sout!(indsa1-ind*256);Sout!val1; Rin!(indsa+ind*256); Rin!val;exit
     []
     [(indsa1<(ind*256)) OR (indsa1>=((ind+1)*256))]-> Rin!(indsa+ind*256); Rin!val;exit
     )
    )
   endproc
endproc


Cent_Per:: process Act_Manager [Min2, Rout, Aout](ind : int): noexit:=
  Min2?indsa:int; Min2?val:int;
   ([(indsa>=(ind*256)) AND (indsa<((ind+1)*256))]->
        Aout!(indsa-ind);Sout!val;Act_Manager [Min2, Rout, Aout](ind);
   []
   [(indsa<(ind*256)) OR (indsa>=((ind+1)*256))]->Act_Manager [Min2, Rout, Aout](ind);
   )
  []
   Rout?indsa; Rout?val; Aout!indsa; Aout!val; Act_Manager [Min2, Rout, Aout](ind);
endproc
```

Fig.4: Control Panel internal processes definition

The sensor Manager, the TTL definition of which is given in fig. 4, essentially has two tasks:

- it transmits the data received from the Master (Min) to the local sensors (Sout);
- it polls the sensors to collect their data (Sin_k).

Fig. 4 gives the TTL definition of the Act Manager.

Each sensor can interact with one or two actuators (e.g. the presence sensor can affect the functioning of both the heating system and the alarm system). The sensor address-actuator address link is given by the Routing Table, which has to be specified

according to the specification of the single control panel. In T-LOTOS this would mean as many descriptions as there are panels. To make the description of the Cent_Per module as generic as possible, we use one of the characteristics of TTL templates - the possibility of using generic names for processes in a given description. In this way there is only one description of the peripheral control panel, which can be personalized every time a panel is instanced giving it the name of the Routing Table process. The latter has the task of sending to the master the data relating to actuators located in other panels, which the master will then send to the proper destination.

3.3 Sensor Controllers

Fig. 5 gives a scheme and TTL definition of the Sensor-Controller module, which controls the sensors. In the instantiation phase the address of each controller is given (ind_s), by which it can be selected to be sent data and the reference value (*val*). In addition, each controller can decide what action the actuator is to take according to difference from the reference value.
Through the input line C_{pin} it is also possible to change the reference value dynamically.

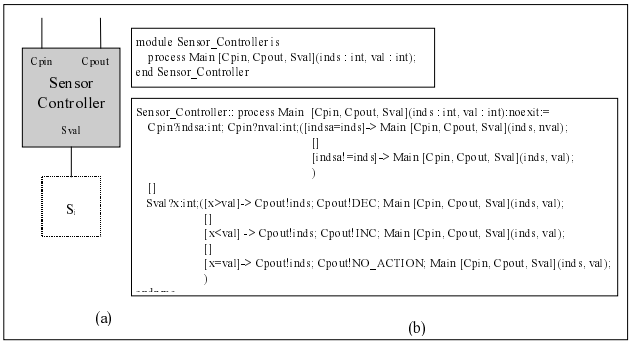

Fig.5: Sensor-Controller schema, its declaration and definition

3.4 Actuator Controllers

Each actuator has a controller which interacts with the Act Manager and the actuator. From the Act Manager it receives data concerning the actions to be performed and transmits it to the actuator. Fig. 6 gives the scheme and TTL definition of the

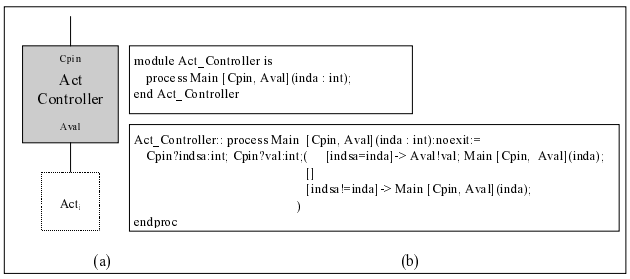

Fig.6: Actuator Controller schema, its declaration and definition

Act_Controller module, which has an 8-bit address (ind_a).

3.5 Example of Application

To show the flexibility of the TTL specifications of the various components, we give an example in which a specific home automation system is designed starting from generic components. It should be noted that, given the way the components are specified, it is possible to design systems with a complex structure.

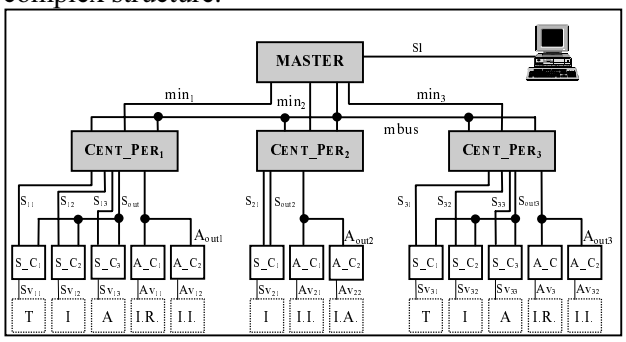

Fig.7: Using modules to build a complete application

Let us assume, for instance, that we want to specify a system with the following features:

- a system which covers three separate rooms;
- for two of the three rooms, control of temperature, lighting and the alarm system;
- for the third room, control of the lighting system alone;
- local management of the heating and lighting system and global control of the alarm system.

```
use comp.dec
specification Home_Automation_System [Sl, Sv11, Sv12, Sv13, Av11, Av12, Sv21, Av21, Av22, Sv31, Sv32, Sv33,
Av31, Av32] : noexit:=
behaviour
    ((((((((((((((Master.Main[Sl, mbus, min[min1, min2, min3]]
    |[min1, mbus]|  Cent_Per.Main[mbus, mbus, min1, Sout1, Sin[S11, S12, S13], Aout1](0)<R_T1[R1in, R1out, min1]>)
    |[min2, mbus]|  Cent_Per.Main[mbus, mbus, min2, Sout2, Sin[S21], Aout2](1)<R_T2[R2in, R2out, min2]>)
    |[min3, mbus]|  Cent_Per.Main[mbus, mbus, min3, Sout3, Sin[S31, S32, S33], Aout3](2)<R_T3[R3in, R3out, min3]>)
    |[S11, Sout1]|  Sensor_Controller.Main[S11, Sout1, Sv11](0, 25))
    |[S12, Sout1]|  Sensor_Controller.Main[S12, Sout1, Sv12](1, 25))
    |[S13, Sout1]|  Sensor_Controller.Main[S13, Sout1, Sv13](2, 25))
    |[Aout1]|  Act_Controller.Main[Aout1, Av11](3))
    |[Aout1]|  Act_Controller.Main[Aout1, Av12](4))
    |[S21, Sout2]|  Sensor_Controller.Main[S21, Sout2, Sv21](0, 25))
    |[Aout2]|  Act_Controller.Main[Aout2, Av21](1))
    |[Aout2]|  Act_Controller.Main[Aout2, Av22](2))
    |[S31, Sout1]|  Sensor_Controller.Main[S31, Sout3, Sv31](0, 25))
    |[S32, Sout1]|  Sensor_Controller.Main[S32, Sout3, Sv32](1, 25))
    |[S33, Sout1]|  Sensor_Controller.Main[S33, Sout3, Sv33](2, 25))
    |[Aout3]|  Act_Controller.Main[Aout3, Av31](3))
    |[Aout3]|  Act_Controller.Main[Aout3, Av32](4))
where
    process R_T1[R1in, R1out, min1] : noexit:=
        R1in?inds:int; R1in?val:int;([inds=0]-> R1out!3; R1out!val; R_T1[R1in, R1out, min1];
                                     []
                                     [inds=1]-> R1out!4; R1out!val; R_T1[R1in, R1out, min1];
                                     []
                                     [inds=2]-> min1!(1*256+2); min1!val; R_T1[R1in, R1out, min1];
                                     )
    endproc
    process R_T2[R2in, R2out, min2] : noexit:=
        R2in?inds:int; R2in?val:int;([inds=0]-> R2out!1; R2out!val; R_T2[R2in, R2out, min2];
                                     []
                                     [inds=1]-> R_T2[R2in, R2out, min2];
    endproc
    process R_T3[R3in, R3out, min3] : noexit:=
        R3in?inds:int; R3in?val:int;([inds=0]-> R3out!3; R3out!val; R_T3[R3in, R3out, min3];
                                     []
                                     [inds=1]-> R3out!4; R3out!val; R_T3[R3in, R3out, min3];
                                     []
                                     [inds=2]-> min3!(1*256+2); min3!val; R_T3[R3in, R3out, min3];
                                     )
    endproc
endspec
```
Fig.8: TTL description of complete system

Fig. 8 shows the main specification of the system which makes use of a library of modules already analyzed, called *comp*, the schema of the system is

given in fig. 7. Of interest is the description of the various Routing Tables, which are passed to the *Cent_Per* module, and the compactness of the description (fig. 8). This is due to the language, because although the number of rooms may vary the reference library does not.

4. Conclusions

TTL has been developed to specify complex hw/sw systems.

It covers many of the gaps left by T-LOTOS, such as the lack of modularization mechanisms, iterative constructs and the capacity to define generic processes.

TTL has been devised in such a way as to be consistent with the object paradigm and further work is being done in this direction.

A TTL specification can be translated into a T-LOTOS one by means of an automatic tool which copes with references to external modules, expands loop constructs and replaces all the template instantiations with the relative T-LOTOS processes. This is a fundamental point because it guarantees the possibility of using all the tools which already exist in T-LOTOS.

In this paper a complete example of application of the TTL to a real case has been presented. Furthermore the example has been analyzed using the tools developed for LOTOS, simply by converting TTL specification into LOTOS one.

References:

[1] ISO IS 8807, *Information Processing Systems, Open System Interconnection, LOTOS, A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour,* ISO, June 1988.

[2] T. Bolognesi, E. Brinksma, *Introduction to the ISO Specification Language LOTOS, Computer Networks and ISD Systems*, 14 (1987) 25-59.

[3] ISO IS 9074, *Information Processing Systems, Open Systems Interconnection, ESTELLE, A Formal Description Technique Based on an Extended State Transition Model*, ISO.

[4] M. Faci, L. Lo Grippo, Specifying hardware systems in LOTOS, *CHDL93*, April 26, 1993

Fig.8: TTL description of complete system

[5] *Manual*, IEEE std 1076-1987.

[6] T. Sternheim, R. Singh, R. Madhaven, Y. Trivedi *Digital Design and Synthesys with Verilog HDL*, Automata Publishing Company, 1993.

[7] A. Bailey, G.A. McCaskill, G.J. Milne An Exercise in the Automatic Verification of Asynchronous Design, *Formal Methods in System Design*, 4, 213-242 (1994).

[8] J. Milne, CIRCAL and the Representation of Communication, Concurrency, and Time. *ACM Transactions on Programming Languages and Systems*, Vol, 7, No. 2, April 1985, pp. 270-298.

[9] G. Berry, G. Gonthier, *The ESTEREL synchronous programming languag: design semantics, implementation.* Science of Computer Programming, 19:87-152, 1992.

[10] V. Carchiolo, M. Malgeri, G. Mangioni, Formal Codesign Methodology with Multistep Partitioning, *VLSI Design Journal*, Volume 7, Number 4, 1998.

[11] V. Carchiolo, M. Malgeri, G. Mangioni, An Approach to the Synthesis of HW/SW Codesign, *Proc. Of 5th IEEE Intl. Workshop on Hardware/Software Codesign*, Braunschweig (Germany), March 24-26, 1997.

[12] J. Quemada, A. Fernandez, Introduction of Quantitative Relative Time into LOTOS, *IFIP Workshop on Protocol Specification Testing and Verification*, VII North Holland 1987.

[13] V. Carchiolo, M. Malgeri , G. Mangioni, TTL: A Lotos Extension for System Description, Proc. Of Theoretical problems on Manufacturing Systems Design and Control, Lisbon, June 1996.

[14] V. Carchiolo, A. Di Stefano, A. Faro, G. Pappalardo, ECCS and LIPS Two Languages for OSI Systems Specification and Verification, *ACM Trans. on Programming Languages and Systems*, Vol.11 No.2, April 1989, pp 284-329.

[15] V. Carchiolo, A. Faro, O. Mirabella, G. Pappalardo, G. Scollo, A LOTOS Specification of the PROWAY Highway Service, IEEE Trans. on Computer, Vol.35 No.11, Nov.1986.

[16] J. Quemda, S. Pavon, A. Fernandez, State explorationby Transformation with LOLA, *Workshop on Automatic Verification Methods for FiniteState Systems*, Grenoble, France, June 1989.