Modelling and Simulation of a Real-Time Hybrid System

Manuel I. Capel-Tuñón

Department of Software Engineering, ETS Informatica y Telecomunicación,
18017 University of Granada, Spain
mcapel@ugr.es

Abstract. A correct system design is systematically obtained from the SA/RT requirements specification model (RSM) of a real-time system. The aim of the systematic procedure is obtaining a complete model in the Matlab/Simulink/Stateflow framework for solving a realistic industrial problem, namely, an AC motor controller which must be able to maintain a constant air flow through a filter. The article also discusses a practical application of the method for implementing a closed loop control system to show how the proposed procedure can be applied to derive complete hybrid system designs.

Keywords: Real-time embedded control systems, SA/RT, Simulink, Stateflow, Process Algebra, System Software Specification.

1 Introduction

Structured Analysis methods for specification of Real-Time systems (SA/RT), applied to the specification of non-functional user requirements, such as timing constraints between system actions, do not address -or excessively postpone- non-functional specification to a final phase of the system development life-cycle, thereby causing economic loses if there are mistakes in the requirement specification phase. Although SA/RT methods help us to find a consistent specification of system requirements, however they must be complemented with other, formal, description methods, which facilitate non-functional specification (i.e., scheduling analysis, resource allocation, timing constraints, etc.) [1, 7-9] on early stages of any target system development.
Firstly, the present contribution is aimed at integrating Stateflow to represent processes of reactive behaviour and Simulink blocks to represent continuous components in the final stages of a real-time system specification that is systematically derived by the application of a set of rules [1]. By using a hierarchy of Stateflow charts as a semiformal graphical description language, we can give an operational semantics to reactive data transformation and control processes that appear in any SA/RT model. In addition, the simulation tools provided by Simulink/Stateflow provide a common framework to carry out a complete system specification of discrete events dynamic systems as well as continuous dynamic systems, see Fig.1.

Different real-time system types require different designs of formal description languages, programming languages and software tools. A series of tools are aimed at modelling real-time systems that integrate continuous components. Among currently implemented approaches, we can distinguish [15] three major classes: (1) block-based, (2) physical oriented and (3) hybrid state machines. (1) Block based tools give a graphical language based on a library of primitive blocks with discrete, continuous or hybrid behaviour. The most used among these tools are: Simulink/Stateflow, Easy5 and VisSim, the latter one being used in iLogix Statemate Magnum. These tools are usually easy to use for building small and medium-size models of target systems, but for complex ones the model becomes a multilevel diagram that is difficult to understand and modify. (2) Physical oriented tools use a system of differential equations to describe the continuous behaviour of an hybrid system; these kinds of tools are mainly academic projects, such as 20-Sim from Controllab Products, Dynasim Dimola and Modelica, Smile from Berlin Technical University, which use a system of differential equations to describe the continuous behaviour of the system; discrete components are difficult to model and to change; this approach works better for modeling pure continuous physical systems. The approach works better for modelling physical systems, but if discrete components are intertwined with continuous ones these tools produce inflexible models, with parameters that are difficult to change at run time in simulations. (3) In hybrid state machines [12] the continuous behaviour described by a system of differential equations associated with the discrete state of a state-transition machine; when the discrete state changes as a result of an event, the continuous behaviour may also change; this approach gives very compact and flexible specifications of complex hybrid systems, but there are very few tools that support this class of tools at the moment, among which are, Path from Berkeley University and Model Vision from Object Technologies.

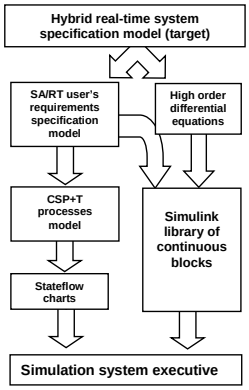

Fig. 1. Software architecture of the proposed model of a real-time system.

CSP+T formal notation is capable of unambiguously describing the different modeling entities of SA/RT notation, which can afterwards be converted into a hierarchy of Stateflow diagrams. A semantic equivalence between CSP+T process terms and a subset of Stateflow modeling entities can be shown, according to our method. The validation of the final system model is carried out by simulation, since

Simulink blocks are very accurate and can be tested in a realistic environment by downloading the target software in an embedded controller, with which the environment can directly communicate through different A/D, D/A interfaces. The imprecision regarding time specification that SA/RT notation presents has been overcome by deploying CSP+T as a meta-notation to label transitions in state-transition diagrams. This way opens up the possibility of having automatic code derivation of the annotated RSM, which can be automatically translated into a Java controller with the support of JCSP [11]. The constructive nature of the method makes it a good candidate to be integrated with off-the-shelf simulation tools for dynamic continuous systems design. This research aims at the implementation of formal tools for performing probably correct automatic generation of code for real-time controllers.

The remainder of the paper is structured as follows. We first give some background on SA/RT modelling methods and CSP+T process algebra. The top-down derivation procedure proposed in this paper is discussed in detail in the next section specifying the steps to be performed. Then, the method proposed is applied to solve an industrial problem of a real-time feedback closed loop used to maintain constant rotor speed of an induction motor driven by a TriaC device such that a constant air flow through a filter in HVAC systems is achieved. The case study shows how the proposed method can be applied to derive a hybrid system that also contains discrete components. The next section describes how the system can be validated by simulation and adds some components to the model. Finally, the conclusions and the ongoing lines of work are presented.

2 Modeling methods

In the proposed top-down derivation method to design a real-time hybrid system with continuous and discrete components, we use a derivation procedure that allows us to obtain a compact specification (CSP+T process terms) of the functional and dynamic aspects (Simulink/Stateflow blocks) of the system.

2.1 Requirements Specification Model (RMS)

A RSM can be obtained by applying a set of SA/RT methods using an informal graphical notation also provided by SA/RT. This model consists of a hierarchy of transformation schemes rooted on the System Context Diagram (SCD). Each scheme "explodes" into a State Transition Diagram (STD) or into a Data Flow Diagram (DFD). The scheme denoted as SCD defines the border between the system, which should be understood as a double model describing the data flow and the control flow relationships in the "solution domain", and the environment, comprising the external entities (or terminators) to the system and representing the "problem domain".

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

CTPs. A transformation scheme is represented by an SCD or by a DFD. DFDs are composed of several copies of the above analysis entities and must include at least one DTP. The bubbles that represent DFDs may "explode" into new, more detailed DFDs. A fundamental strategy of SA/RT is to separate the control and the data process descriptions within the system. A CTP is formally specified by means of a state transition diagram (STD). STDs should be deterministic Moore or Mealy automata, and they describe a sequence of state transitions of the system that cause the execution of DTPs to be triggered. The SCD of the constant air flow through a filter control system can be seen in Fig.4.

2.2 Flaws of SA/RT as a Specification Notation for Real-time Systems

The following ambiguities appear in both the WM [2] and HP [3] notations, causing imprecisions in the specification, and therefore non predictability may be present [9] in final real-time systems at a later development stage:

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

b) The enabling conditions of processes are not fixed. The SA rationale is that processes are enabled whenever "sufficient data" appear in any of their input flows. However, the enabling conditions rules do not clearly indicate the expected behaviour of a process when more than one of its input flows are carrying values. In that case, a non-deterministic selection appears in a process execution sequence and there is no SA entity foreseen to represent it.

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

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

e) Simultaneous events awakening more than one transition. This possibility is excluded in SA/RT notations since transitions exiting the same state are associated with different events, since STDs are Mealy machines. However, there should be no objection to allowing nondeterministic selection of transitions in notations for soft real-time systems. Many proposals have been made in the last years to overcome the problem of SA imprecision by complementing a system specification with formal methods. The use of extensions of algebraic process description languages [7], such as CSP[17], CSP+T [4], or the standard specification language LOTOS, can give a precise and flexible interpretation to SA entities. In this respect, it has been shown [1] that CSP+T process algebra formalizes the semantics of a SA/RT specification model and also allows for the specification of timing constraints between the occurrences of actions during any execution of the system by using a defined set of rules.

2.3 Real-time systems specification with CSP+T

The group of CSP derivatives to describe time intervals includes Timed CSP [17] and CSP+T, the latter being a simpler approach. Providing less descriptive power, although still powerful enough to formally describe a set of primitive processes with time constrained behavior, CSP+T is an adequate formal specification language for the majority of real-time systems.

The syntax of CSP+T, adapted to our method, which is detailed in [1] is next described,

−Every process P defines its own set of communication symbols, i.e. its finite communication alphabet $\alpha(P)$. These communications represent the events that the process P receives from its environment or internally occur (e.g. the null action τ). Any type of event causes a change of state of the process. Internal events, such as τ , are not externally visible.

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

−An instance of a process term must be created before it can execute. Thus, an operator, $\star$ (star) denotes process instantiation. This event is unique in the system since it represents the origin of a single global time line in the system to which the execution time of each process refers. Let us consider a process P that initially can only engage in the event a. Given P', the timed version of P, which is instantiated at time 1, and that s is a time stamp associated to a, the specification of P' becomes

$$P' = 1.\ \star \rightarrow s.a \rightarrow STOP,\ where\ s \in [1, \infty) \qquad (1)$$

−An event operator $\triangleright\triangleleft$ is introduced to be used jointly with a variable to record the time instant at which the event occurs, so that ev $\triangleright\triangleleft$ v means that the time at which ev is observed in a process execution is recorded in the variable v. As several successive events can instantiate the same variable at different times, if we specify the process,

$$P = 1.\ \star \rightarrow a \triangleright\triangleleft var \rightarrow STOP \qquad (2)$$

For each process execution, *var* will record the corresponding value of the time at which the event *a* occurred, and it will always satisfy $var \geq 1$. The variables associated to the operator $\triangleright\triangleleft$ are called marker variables and their scope is strictly limited to one sequential process.

−Each event is associated with a time interval, which is called the event-enabling interval. This interval represents the period of time during which the event considered is available to the process and its environment, and is relative to some preceding event of the current process execution. A process is considered to be the STOP process if it cannot engage in an alternative within the enabling interval of the event. The event-enabling intervals are continuous. Let us suppose, for instance, that a process P can only engage in event a, which can only occur between 1 and 2 units of time from the instantiation time, recording itself in the marker variable v as the time at which this event has occurred:

$$P = 0.\ \star \rightarrow [1,2]\ a \triangleright \triangleleft v \rightarrow STOP. \qquad (3)$$

- The value of the marker variable v will satisfy the inequality $1 \leq v \leq 2$. The enabling intervals are defined in terms of functions, as $rel(t_i, v_i) = [v_i, t_i+v_i]$, over a set of marker variables $\{v_i\}$. When there are no marker variables referenced, the enabling interval is defined relative to the immediately preceding event.

$$P = ...\ E.P' .\quad E = \{s \mid s = rel(x, v)\} \qquad (4)$$

–If the preceding event occurs at time t_0, then $rel(x, v) = [v-t_0, x+v-t_0]$, since the times for events are absolute and the times for intervals are relative to the preceding event. The semantics of the parallel composition of two processes with enabling intervals depends on whether the values of these intervals are identical, partially overlapping or disjoint. In the case of disjoint intervals, the parallel composition behaves identically to the STOP process.

2.4 Generation of a System Specification from the RSM

In order to obtain a model of the system, it is necessary to represent every analysis entity of the RSM by a class of CSP+T processes. Following this approach, we write a CSP+T process prototype for every DTP, CTP, DS, CS, etc. A series of transformation rules [1] allow us to create a process term of the algebra for every transformation scheme that appears in any diagram of the RSM. A representation of the complete derivation procedure is shown in Fig 2. The modelling elements of Stateflow diagrams can be represented by CSP+T terms, as reactive processes at the lowest level of a system specification. As Simulink blocks are the basic bricks needed to represent primitive functions, as well as continuous components in many hybrid real-time systems simulations, we replace the primitive processes in a RSM model by Simulink blocks in order to get an integrated model of a hybrid real-time system that can afterwards be validated by simulation.
To carry out a simulation in the Simulink/Stateflow framework, additional blocks must be added to the final model. These blocks represent the external entities of RSM and must be modeled according to the specific physical devices (actuator or sensor) that supply signals (data or event) to/from the system.
In general, we will adopt a bottom-up process that consists of the following steps:
1) Prepare the analysis schemes for carrying out the transformation. It may be necessary to rename some analysis entities to avoid conflicts, i.e., unwanted synchronizations between processes.
2) Transform the control transformation schemes (CTP) and data transformation schemes (DTP) that present reactive behaviour of the lower level, i.e. those that do not explode into other schemes, into Stateflow diagrams.
3) Add Simulink-blocks to represent external devices or continuous components, i.e., when they are needed to represent DTPs with transformational behaviour .

4) Select the other schemes, i.e., Data Storage (DS), Control Storage (CS), DTP, CTP, or Continuous Flow of Data, that appear in the scheme, in ascending order; and build a CSP+T process for each entity in the scheme.
5) Once the CSP+T model has been obtained for all the entities in an SA/RT scheme, one CSP+T process will be defined to model the complete scheme. If this scheme is already included in a CTP or a DTP of a higher level, repeat step (4), thus progressively integrating the CSP+T model of the system in an ascending way.
The iterative process finishes when a unique process with the communication interface of the system is obtained, i.e., when the CSP+T model of the System Context Diagram is obtained.

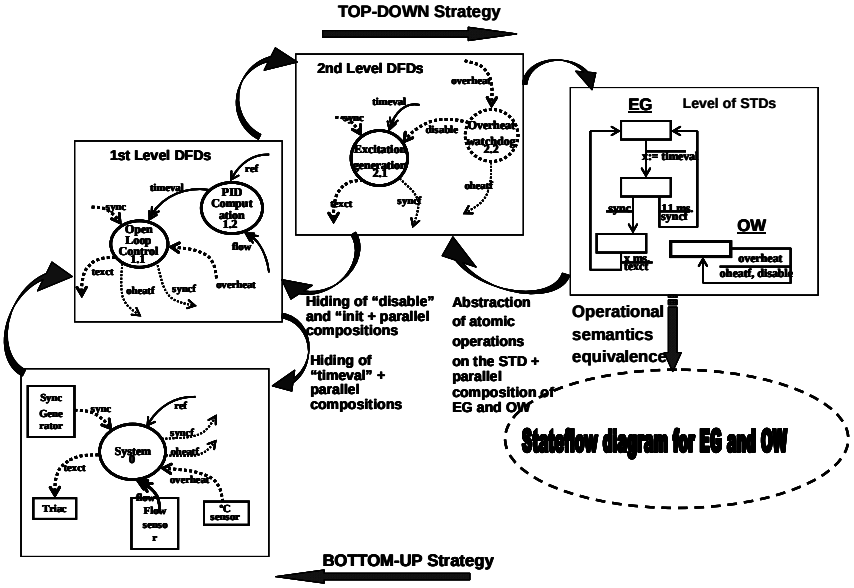

Fig. 2. The complete derivation procedure of a real-time system

3 Regulation of Rotor Speed With An Induction Motor

3.1 Description of the Induction Motor Control

An informal description of the user´s requirements specification of a closed loop control system is presented for controlling an AC motor (or induction motor), Fig.3. The open loop control of the engine is obtained by feeding it with a controlled voltage of 220 volts and 50 Hz. This control is carried out by cutting the sinus wave, which represents the input voltage using an electronic device named TriaC, which operates as a very fast switch.

The control line of the TriaC is driven by a synchronization signal (synch), which informs when the input voltage passes through a zero value, at this moment the TriaC automatically stops to conduct electricity. If after switching the TriaC off, it is fired a number of milliseconds later, it will be driven to saturation by the signal texct and will start to conduct until the input voltage passes through a zero value again. The closed loop of control is obtained in this case by calculating the precise time at which the TriaC must be fired, so the excitation time must be calculated in real-time and in every cycle of the input voltage. The system should address its own safety if synchronization signal fails or TriaC overheats. If synch is missed after passing a complete cycle of the input voltage, then syncf is raised. Other possible failure could happen if the TriaC overheated, in this case the electronic device might short-circuit and lead the engine to start working at the maximum number of revolutions, which would cause the loss of the engine after 1 second approximately.

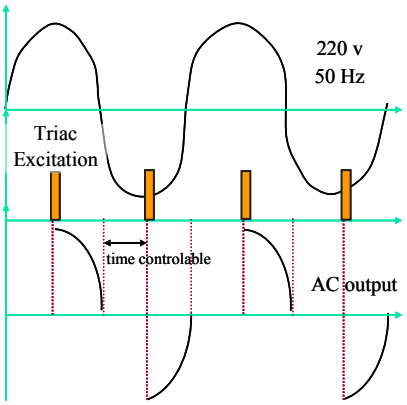

Fig. 3. Description of the operation of TriaC device that controls the AC induction motor.

The combination of induction motor with TriaC device can be used to control or maintain constant the velocity of a centrifuge of washing machine, the air flow through a filter, the speed of a vehicle, etc. From now on we suppose that AC induction motor directed by the TriaC device controls the air flow through a filter.

3.2 Top-down derivation of the System Model

A set of hierarchical diagrams is obtained using top-down strategy within the SA/RT methodology. The high-level diagram is the System Context Diagram (SCD). The SCD (fig 4) includes 5 control flows: the synchronization signal (synch), which informs when the input voltage passes trough zero value; the TriaC overheating warning; two signals, the first one signals the missing of synch and the second one signals TriaC overheating; the excitation (texct) signals to make the TriaC return to allow current to pass again. It also includes 2 data flow: the first one gives the present air flow trough the filter (flow) and the second one, the air flow reference value (ref). Note that the automatic system modeled interacts with external devices that supply data flows to the system like sensors, or interacts with external devices sending

control events which change the state of system environment like actuators. Then the AC motor is not controlled directly by the system, consequently it is not considered as an external entity of the system. The latter one computes a timeval value that activates the excitation signal of TriaC texct. This device causes an immediate effect to the induction motor by changing the actual speed of motor which can only be made available to the system by sensors as tachometers.

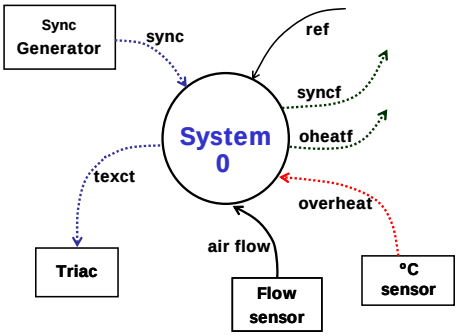

Fig. 4. System Context Diagram.

The SCD "explodes" into two main DTP processes on the 1st level DFD: Proportional-Integral-Derivative control (PID) and the open loop control (OLC). The OLC process triggers signals to actuators depending of the input data flows; i.e. textct from an input timeval value, oheatf from an positive overheat value, syncf when the synchronization signal is missed after passing one complete cycle of the input voltage. However the PID control process determines the correct time (timeval) at which the TriaC must be fired within the present voltage cycle. The integration of two DTPs represents in fact the Closed Loop Control (CLC) of the system. Once the PID control process adjusts the timeval value according to the actual speed of motor, the OLC process produces a new excitation signal that changes the actual speed until the actual speed reaches the reference speed.

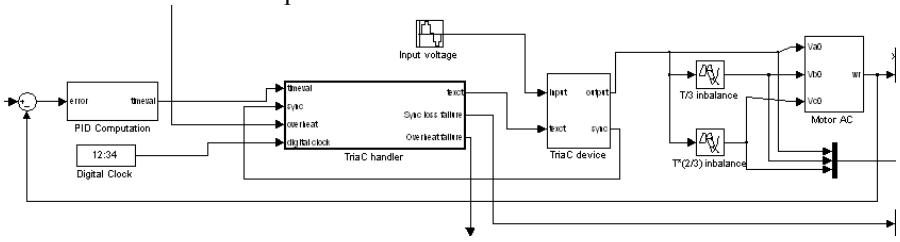

Fig. 5. Simulink model of the final system.

A static analysis of the system reveals that the shorter the timeval is the higher the speed reached by the rotor will be; but the relationship between these variables is non-linear. The corrected timeval has been obtained as follow. First, the PID control computes a positive or negative timeval increment which is added to the previous

value of timeval depending on whether the actual speed is over or under the speed reference value, respectively. If the timeval is outside the interval [0,1/freq*2], its value is saturated by the maximum or minimum value. Then one Simulink subsystem block is designed as the PID controller with its interface made up of two ports: the error signal (speed error signal) as the input port and the corrected timeval as the output one.

4 Simulation of The Hybrid System

In order to carry out a simulation of the hybrid system proposed in this paper, more components must be added to the model. We need to construct one model of a TriaC device, one AC Motor , the flow sensor (or a sensor to measure the speed of the rotor like a tachometer), the temperature sensor and one sync generator. This model can be implemented in Simulink/Stateflow framework as Fig. 5 shows.

4.1 Physical Modeling of an Induction Motor

The most difficult component to model is the AC Motor, since the rest of them can be modeled by means of simple switches or a combination of them. The following sections discuss the model of AC motor developed for the system.

The functioning of an induction motor is based on the Physical principle of mutual induction between electrical circuits traversed by a variable magnetic flux Φ. According to the Faraday law, which is given by the following equation, $\varepsilon = -\frac{d}{dt}(N \cdot \Phi_B)$, the magnetic flux traversing a motor winding only depends on the current conducted by the circuit. It does not depend, for instance, on the number of poles of the motor. We can assign a self-induction constant L to any circuit being affected by magnetic induction, according to the equation $N \cdot \Phi_B = L \cdot i_{reel}$

Nowadays the winding of induction motors is made of three windings, carrying each one of them a voltage phase separated $2\pi/3$ rad. from the next phase, which yields a rotating magnetic field in the stator, Fig.12. The velocity of rotation is called the synchronous speed, which is given as a parameter of induction motors. If we short-circuit the rotor winding –using a squirrel cage winding, for instance-, then the motor will start rotating because the change in the magnetic field direction yielded by the synchronous speed of the stator ωe induces a current that produces an electromagnetic force in the rotor. The difference between the rotation velocity of the stator ωe and this one of the rotor ωr is named slip, which is also given as a parameter of induction motors.

4.2 The two-phase synchronous rotating frame

We can assume a reference system that rotates at the synchronous speed ωe of the stator to ease the representation of the rotating $\vec{B}$ and inductance linkages by a system of coupled differential equations that describe the physical induction and motor dynamics, as Fig. 6 shows. The induction motor is therefore modeled by two reels, the first one is aimed at conducting the current in the stator and it also generates the rotating magnetic field. The second one generates the induced magnetic field in the rotor. This simple model allow us to describe the magnetic coupling between the stator and rotor windings of an induction motor very accurately.

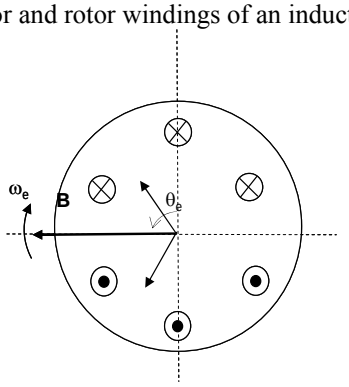

Fig. 6. Motor winding.

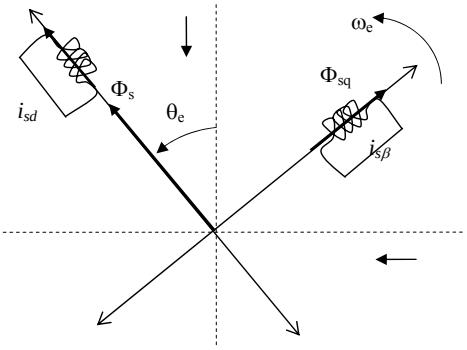

Fig. 7. Rotating reference system

The first axis, Fig. 7, is called the *direct axis* and the second one, the *quadrature axis*. θ_e is an additional variable representing the rotor angle and it can be considered an additional state of the induction motor model. An induction motor with a *squirrel cage* rotor winding (short-circuited) will have null v_{qr} and v_{dr} voltages. The constants and system variables of the above linear differential equations are given in Table 1.

Table 1. Constants and variables of the physical variables of an induction motor.

Variables of Physical Model	
d: direct axis of the rotating reference system	χ^*_{lm}= 1/(1/χ_{ls}+1/χ_{lr}+1/χ_m): total reactance with the loses for magnetizing (χ_m)
q: quadrature axis of the rotating reference system	i_{qs}, i_{ds}*:* currents of the q and d stator axis
s: subindex for the stator variable	i_{qr}, i_{dr}*:* currents of the q and d rotor axis
r: subindex for the rotor variable	*p*: number of poles of the motor
Fij=*Φij*, where *i*=*q* or *d* and *j*=*s* or *r*, magnetic linkage	*J*: inertia momentum
v_{qs}, v_{ds}*:* stator voltages	M_e: motor electrical torque (output variable)
v_{qr}, v_{dr}*:* rotor voltages	M_l: load torque (input variable)
R_r, R_s*:* rotor and stator resistors	ω_e: stator synchronous speed (input variable)
χ_{ls}: stator reactance ($\omega_e L_{ls}$)	$\omega_b=2\cdot\pi\cdot f_b$: angular speed corresponding to the

electric frequence of the motor feeding voltage.

4.3 Calculation of the electromagnetic torque generated by the motor

We can derive a mathematical expression for calculating the electromagnetic torque generated by the motor. Since we know that the mechanical power given by the motor is obtained from the electromagnetic equation $P_{mech} = \frac{3}{2}\left(\omega_b \cdot \Phi_{sq} \cdot i_{sd} - \omega_b \cdot \Phi_{sd} \cdot i_{sq}\right)$, and the magnetic linkage $\Phi_{sq,sd}$ only depends on the synchronous angular speed and the magnetic flux Fij = $\omega_e \cdot \Phi_{ij}$. The mechanical power can be made equivalent to the electrical torque Te generated by the motor, then we can obtain $T_e = \frac{3}{2}\left(\frac{p}{2}\right)\frac{1}{\omega_b}\left(F_{ds} i_{qs} - F_{qs} i_{ds}\right)$ from the magnetic linkages F_{ds}, F_{qs}, and currents, which are obtained by solving a system of differential linear equations with concrete values of p (the number of poles) and ω_e as the input data to the induction motor model. The angular velocity ω_r of the rotor can also be calculated, since the load torque T_l and moment of inertia J are also parameters of the induction motor model. As the above equations show, the electrical torque and the angular rotor velocity depend on the number of poles of the rotor winding, on the contrary of what happens with the magnetic couplings.

5 Matlab/Simulink model of an induction motor drive

The model of an induction motor http://lsi.ugr.es/~mcapel/miscelanea/motor has been structured in 3 main blocks: (1) transforms the three stator voltages v_a, v_b, v_c , with a phase of $2\pi/3$ between each two, into the rotating reference system *dq*; (2) the block representing the induction motor itself (which inputs the three phase voltages, the synchronous angular speed ω_e and the load torque); (3) this block returns the expression of the model variables in the *dq* system back to the three phases abc reference system, since the latter one give us the standard graphical representation of currents in the stator.

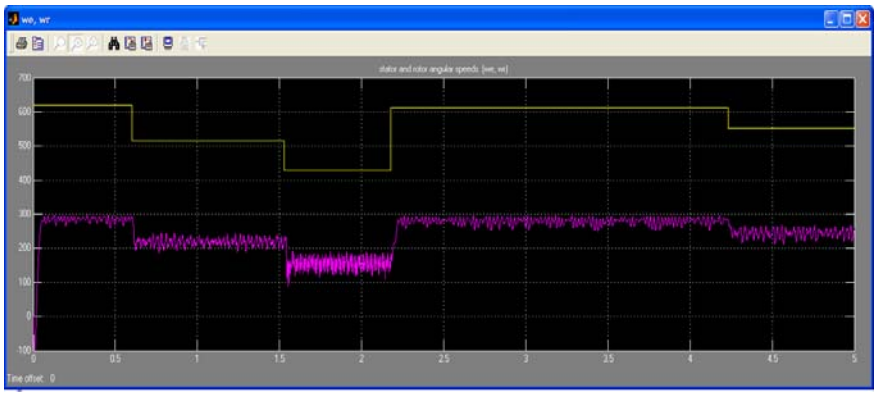

Fig. 8. Response of the rotor to changes in the stator speed

Two specific blocks have been designed to calculate the electrical torque Me given by the motor and another to calculate the rotor axis angular speed ω_r.

Finally, all the physical model constants given in table I have been defined using IS physical units in a m-file of Simulink, which has to be executed in Matlab before opening the Simulink model of the system.

6 Obtained Results

The results obtained with the two models (OLP and CLP) were quite different. In the first case, it was only considered an open control loop model; thus, only after a constant time the *TriaC* is fired in every cycle. In this case the disturbances in the system response (ω_r) are remarkable. Rotor speed follows the changes produced in the synchronous angular speed in the stator (ω_e), but any change in the value of ω_e provokes fast oscillations around the new value in the rotor velocity, see Fig. 9. If we carry out a simulation with the rotor velocity controlled by a PID, then we will obtain better results.

Moreover, if we take a plot of the electrical torque output by the induction motor w.r.t. a constant load torque, which is given as an input variable to the system, we will take only important oscillations at the beginning, while the system is trying to get a stabilisation point. The oscillations shown in Fig.8 represent about the 20% of the target value for the torque M_e, (300 Nm),; these oscillations are caused by dynamic conditions during motor functioning, as the rotor axis friction.

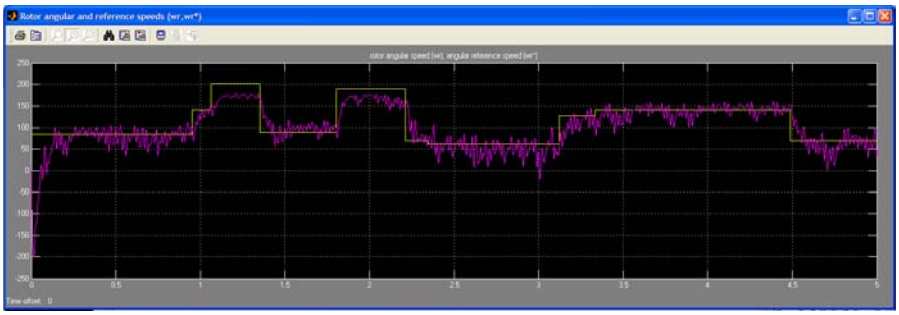

Fig. 9. Closed loop control model response to changes in the stator speed

7 Conclusions

We have presented a derivation procedure to obtain a correct system specification from a semi-formal SA/RT system requirements specification of a given real-time system. The imprecision and ambiguities intrinsic to SA/RT notations have been overcome in our method by using a formal description language based on CSP+T process algebra. Simulink/Stateflow and its library of blocks, which are of use for modeling continuous and discrete dynamic systems, have been used to integrate continuous components in a hybrid real/time system design. However, unlike other proposals that attempted to overcome the same problem by complementing SA with formal methods, our methodological scheme is mainly a set of guidelines, which have proved to be of use for deriving a verifiable specification model of complex systems, as in the case of the application case discussed in the article: the induction motor drive, and controlling an AC motor to maintain a constant air flow through a filter. The method has been defined for its easy integration in ASE environments and/or formal tools based on SA notations. We are currently working on the development of a formal software tool based on JCSP [11] and Java, and which is capable of automated specification, verification and code generation of real-time and embedded system software for several computing platforms.

References

1. Capel, M.I., Holgado, J.A., Balsas, J.R.: A Transformational Approach to the Systematic Design of Real-time Systems. Manufacturing Eng., 3, 2, 5--13, (2004).
2. Ward, P.T., Mellor, S.: Structured Development of Real-Time Systems. Prentice-Hall, Englewood Cliffs (N.J.), (1985).
3. Hatley, D.J., Pirbhai, I.A.: Strategies for Real-Time Systems Specification, Dorset House, New York, (1988).
4. Žic, J.J.: Time-Constrained Buffer Specifications in CSP+T and Timed CSP. ACM Transactions on Programming Languages and Systems, 16, 6, 1661—1674, (1994).
5. Demarco, T: System Analysis and Specification, Yourdon Press, (1971).

6. Semmens, L.T., Allen, P.M.: Using Yourdon and Z: An Approach to Formal Specification. In: Z User Workshop, Oxford, UK, 228—253, (1990).
7. Fencott, P.C., Galloway, A.J., Lockyer, M.A., O'brien, S.J., Pearson, S.: Formalising the Semantics of Ward-Mellor SA/RT Essential Models Using Process Algebra. In: FME'94: Industrial Benefit of Formal Methods. LNCS 873, Springer-Verlag, 681—702, (1994).
8. Elmstrom, R., Lintualampi, R., Pezze, M.: Giving Semantics to SA/RT by Means of High-Level Timed Petri Nets. J. Real Time Systems, 5, 2/3, 249—271 (1993).
9. Baresi, L., Pezzè, M.:Towards Formalising Structural Analisis. ACM Transactions on Software Engineering and Methodology, 7, 80-107 (1998).
10. Structured Technology Group: AxiomSys/AxiomDsn: Design CASE Tool. Structured Technology Group. Saugus, California, USA, (1995).
11. Welch, P: Process Oriented Design for Java: Concurrency for All. In: Parallel and Distributed Processing Techniques and Applications, PDPTA 2001, USA, (2001).
12. Maler, O., Manna, Z., Pnueli,A.: From timed to hybrid systems. Proceedings of REX workshop "Real-time: theory in practice", Springer-Verlag, (1992).
13. Harel, D., Gery, E: Executable object modeling with Statecharts. Computer, 30, 7 (1997).
14. Alur, R. Et Al. : Modular specification of hybrid systems in Charon. Proceedings HSCC 2000. LNCS 1790, 6—19 (2000).
15. Borschev, A., Koleshov, Y., Senichenkov, Y.: Java Engine for UML based hybrid state machines. Proceedings of the 2000 Winter Simulation Conference, 1888—1894 (2000).
16. Elmqvist, H., Mattson, E., Otter, M.:Modelica-the new object oriented language. Proceedings 12th European Simulation Multiconference, 127-131 (1998).
17. Hoare, C.Ar.: Communicating Sequential Processes, Prentice-Hall, (1985).
18. Rational Rose Real Time, http://www.rational.com.