A New Design Procedure for a Real-Time Hybrid System Model

Manuel I. Capel, Juan A. Holgado

Departamento de Lenguajes y Sistemas Informáticos.
ETSII Ingeniería Informática. Universidad de Granada,
18071 Granada, Spain
{mcapel, jholgado}@ugr.es
http://lsi.ugr.es/~sc

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

Keywords: *Hybrid systems, formal analysis tools, modular design, process algebras, CSP+T, real-time systems , embedded control systems.*

1 Introduction

Structured Analysis methods applied to the specification of Real-Time systems (SA/RT) are intended to integrate the requirements specification, based on Tom De Marco [1] SA/SD methods, with finite state machine theory, needed to tackle the complex control structures that real-time embedded systems present. These methods consist of a set of techniques and modelling tools that are useful to clarify the user requirements of a system and to produce a requirements specification model (*RSM*) that views the system from two aspects: (1) the *functional* view; (2) the control *reactive behaviour*, i.e., how the system state changes over time in response to external stimuli and internal signals.

Different real-time system types require different designs of formal description languages, programming languages and software tools. In the case of *continuous dynamic systems*, we can distinguish [2] three major classes of software tools:

(1) Block-based, being based on a library of primitive blocks with discrete, continuous or hybrid behaviour, are usually easy to use for building small and medium-size models of target systems, but, for complex ones, yield systems models difficult to understand and modify. The most often-used among these tools are: Simulink/Stateflow, Easy5, and VisSim, the latter used in iLogix Statemate Magnum.

(2) Physical oriented tools which use a system of differential equations to describe the continuous behaviour of a system; discrete components are difficult to model and

to change at run time simulations. These tools are mainly academic projects, such as 20-Sim from Controllab Products, Dynasim Dimola and Modelica, Smile from Berlin Technical University.

(3) Hybrid state machines in which the continuous behaviour is described by a system of differential equations associated with the discrete state of a state-transition machine; there are very few tools that support hybrid state machines at the moment, but this line is gaining momentum now. Some examples of this type of tools are Path from Berkeley University, and Model Vision from Object technologies.

There are other models and software tools based on Petri nets [3] aimed to represent and to verify embedded systems, but not continuous components.

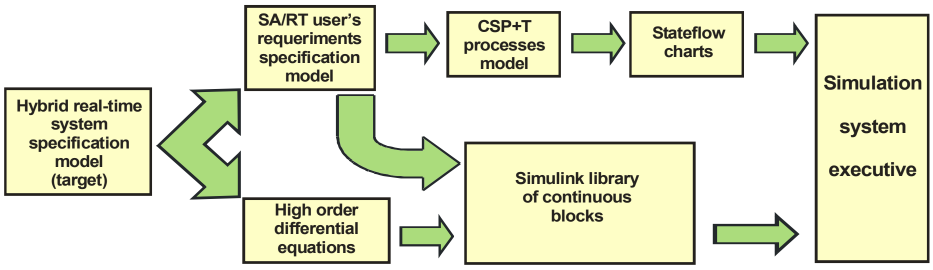

Hybrid real-time system specification model (target)
SA/RT user's requeriments specification model
CSP+T processes model
Stateflow charts
Simulation system executive
High order differential equations
Simulink library of continuous blocks

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

Our method is intended to help with the specification of complex hybrid systems by starting bottom-up from an *SA/RT* specification model and allows integrating blocks describing continuous behaviour within a common methodological infrastructure, since it allows the integration of Simulink/Stateflow blocks at the lower level of a target *RSM* without affecting the behaviour of the other components. The proposed procedure is intended as a new derivation method for obtaining the complete design of real-time systems in Simulink/Stateflow by giving an operational semantics in terms of *CSP+T* process algebra to the semi-formal *SA/RT* analysis entities, Fig.1. *CSP+T* formal notation is capable of unambiguously describing the different modelling entities of *SA/RT* notation, which can afterwards be converted into a hierarchy of Stateflow diagrams, according to our method, since a semantic equivalence between *CSP+T* process terms and a subset of Stateflow can be shown. The transformation of *SA* entities into process terms is carried out by a system of rules, introduced in a previous work [4].

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 latter application of the method opens up the possibility of deploying it to carry out automatic code generation for different target platforms.

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 a 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, adds some components to the model, and discusses results obtained until now. Finally, the conclusions and the ongoing lines of work are presented.

2 Modeling Methods

This section describes the modelling methods used in the proposed method to design a real-time hybrid system with continuous and discrete components.

2.1 Requirements Specification Model (RSM)

An *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, and the environment, comprising the external entities (or *terminators*) to the system.

A transformation scheme in *SA/RT* notation 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*. *DTPs* are schemes of the lower level in a *DFD,* i.e. those that do not "explode" into other schemes, which change the input flows into output flows with no relation between the number of inputs and outputs. The same output can be sent to several analysis entities. A *DTP* must have at least one output flow. The *bubbles* that represent *DFD*s may explode into new, more detailed. *DFDs*. *CTPs* are schemes of the lower level that serve to transform input into output *event flows*. They cannot accept or generate any type of data flow, since 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*), which should be a deterministic Moore or Mealy automaton. Each transition comprises a *condition* (represented by an input control flow in the *DFD*) and an *action* that includes the activities to be carried out before the system reaches a new state. *Data stores* (*DS*) loosely represent data of a certain type that cannot be considered structured. *Control Stores* (*CS*) can only store events of the same type, which are queued in *FIFO* order. Unlike the *DS* semantics*,* reading an event of a *CS* is a destructive operation.

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

Some *ambiguities* appear in current *SA/RT* notations [5, 6], causing imprecision in the specification, and therefore non-predictability, when they are used in real-time systems at a later development stage [7]. Consequently, many proposals have been made to overcome the problem of *SA* imprecision by complementing a system specification with formal methods in the past years. The use of extensions of algebraic process description languages [8], such as *CSP* [9], *CSP+T* [10], or the standard specification language *LOTOS*, can give a precise and flexible interpretation to *SA* entities. In this respect, it has been shown [4] that *CSP+T* process algebra formalizes the semantics of an *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 System Specification with CSP+T

The group of *CSP* derivatives to describe time intervals includes *Timed CSP* [9] and *CSP+T* [10], 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 behaviour, *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 [11], is as follows:

- Every process *P* defines its own set of communication symbols, named *communication alphabet* $\alpha(P)$. These communications represent the events that the process *P* receives or internally occurs, such as the event τ that is not visible in the environment of the system.
- 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 operator, $\star$ (star) denotes process instantiation. Given *P'*, the timed version of *P*, which is instantiated at time 1, the specification of *P'* becomes

$$P' = 1.\star \rightarrow s.a \rightarrow STOP, \text{ where } s \in [1,\infty)$$

This event is unique in the system since it represents the origin of time at which the processes can start their execution.

- An event operator $\rhd\lhd$ to be used jointly with a variable to record the time instant at which the event occurs $ev \rhd\lhd v$ means that the time at which *ev* is observed is in *v*.

$$P = 1.\ \star \rightarrow a \rhd\lhd var \rightarrow STOP$$

For each execution of *P*, the time at which *a* occurred will always satisfy $var \geq 1$.

- Each event is associated with a time interval, called the *event-enabling interval.*

$$P = 0.\ \star \rightarrow [1,2]\ a \rhd\lhd v \rightarrow STOP$$

The value of the *marker variable v* will satisfy the inequality $1 \leq v \leq 2$. Only during this continuous time interval is the event available to the process and its environment. A process is considered to be the *STOP* process if it cannot engage in any communication or synchronize in any event within the interval that precedes the event.

- If the preceding event occurs at time t_0, then $rel(x, v) = [x+v-t_0,\ v-t_0\ ]$, since the times for events are absolute and for intervals are relative to the preceding event.

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

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 [4] allow us to create a process term of the algebra for every transformation scheme that appears in any diagram of the *RSM.*

Thanks to the compositional nature and the capability of Stateflow charts to represent the reactive behaviour of processes, a semantic equivalence between valid *CSP+T* process terms and a subset of the modelling elements of Stateflow diagrams can be established, and we make use of this equivalence to model reactive processes at the lowest level of a system specification. The system specification model is complemented with Simulink blocks that are the basic bricks needed to represent primitive functions, as well as continuous components in many hybrid real-time systems simulations.

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 modelled according to the specific physical devices (actuator or sensor) that supply signals (data or events) to/from the system.

The interactions that a *DTP* or a *CTP* can provoke within their environments by discrete *flows* are modelled as occurrences of events by means of *CSP* synchronous communications. However, continuous flows need to be modelled by an extra intermediate process, *SYNC,* that provides a non-blocking continuous reading/writing from/to a continuous flow as follows,

$$SYNC= flow\ ?\ x \rightarrow S;\ S=flow\ ?\ x \rightarrow S \mid flow\ !\ x \rightarrow S;$$

In general, to derive a Simulink/Stateflow model from an *RSM* one, we will adopt a bottom-up process that consists of the following steps:

1) Prepare the analysis schemes for carrying out the transformation. Some flows in the *RSM* may need to be renamed to prevent unwanted synchronizations in the *CSP+T* semantic equivalent model.

2) Transform the process terms representing control transformation (*CTP*) and data transformation (*DTP*) schemes that present *reactive behaviour* of the lower level into Stateflow diagrams.

3) Add Simulink blocks to represent external devices or continuous components.

4) Select the other modelling entities, such as *DS*, *CS*, *DTP*, *CTP* schemes, or *Continuous Data Flows,* 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 from step (4), thus progressively integrating the *CSP+T* model of the system in ascending order.

The iterative process finishes when a unique process with the communication interface of the system context diagram of the initial *RSM* is obtained.

3 Regulation of Rotor Speed with an Induction Motor

An informal description of the user requirements specification of a closed loop control system is presented for controlling an AC (or *induction*) motor, Fig.2(a). 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, and at this moment the *TriaC* automatically stops conducting electricity. If after switching the *TriaC* off, it is excited a number of milliseconds later, it will be driven to saturation by the signal *texct* and will start to conduct until the input voltage again passes through a zero value. The closed loop of control is obtained in this case by calculating the precise time at which the *TriaC* must be excited, so the excitation time must be calculated in real-time and in every cycle of the input voltage.

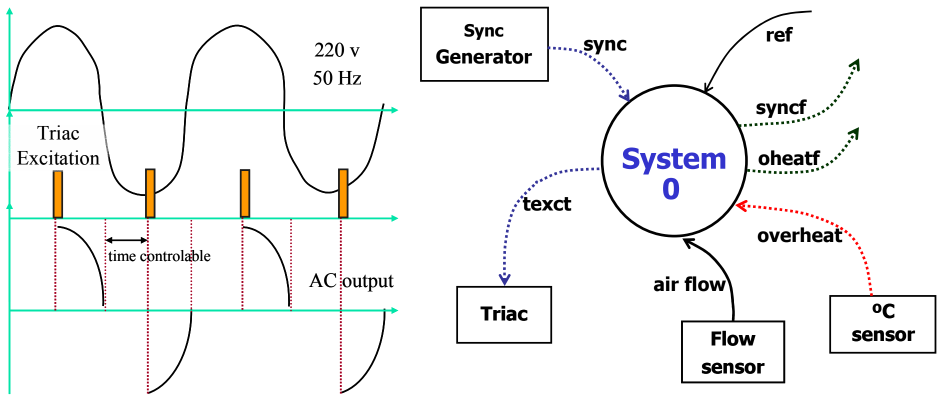


Fig. 2. (a) Operation of TriaC device controlling an AC motor. (b) System Context Diagram

The system should address its own safety if the synchronization signal fails or if *TriaC* overheats. If *synch* is missed after passing a complete cycle of the input voltage, then *syncf* is raised. Another possible failure could occur if the *TriaC* overheated; in this case, the electronic device might short-circuit and cause the engine to start working at the maximum number of revolutions, which would lead to the loss of the engine in approximately 1 second.

The combination of an induction motor with the *TriaC* device can be used to control or maintain a constant velocity, such as in the centrifuge of a washing machine, the air flow through a filter, the movement of a vehicle, etc. We apply the method to obtain a system to maintain the air flow velocity constant through a filter.

3.1 Top-Down Derivation of the System Model

The System Context Diagram (SCD), Fig. 2(b), includes 5 control flows: the synchronization signal (*synch*), which informs us when the input voltage passes through a value zero; the *TriaC* overheating warning; two signals, the first one warning of the missing of *synch* and the second one of *TriaC* overheating; and the excitation (*texct*) signal to make the *TriaC* return to allow current to pass again. It also includes 2 data flows: the first one gives the present air flow through the filter (*flow*) and the second one, the air reference value (*ref*).

The automatic system modeled interacts with external devices that supply data flows to the system –sensors-, or interacts with external devices sending control events –actuators-. The AC motor is not directly interacted by the system, but through the *TriaC* device, and consequently it is not considered an external entity. The system computes a *timeval* value; once this time is elapsed, the system activates the excitation signal *texct* that immediately changes the current value of the induction motor speed. A new value of the speed can be read again by the system through sensors as tachometers; in the present example, the system obtains the updated speed value from one air velocity flow sensor.

The *SCD* "explodes" into two main *DTP* processes on 1st level *DFD*: Proportional-Integral-Derivative control (*PID*) and open loop control (*OLC*), Fig.3(a). The *OLC* process triggers signals to actuators, i.e., *texct* from an input *timeval* value, *oheat* from a positive overheat value. The air flow reference value (*ref*), together with the present air flow value (*flow*), serves as input data to a *Proportional Integral Derivative (PID)* control algorithm to determine the correct time (*timeval*) at which the *TriaC* must be excited within the present voltage cycle.

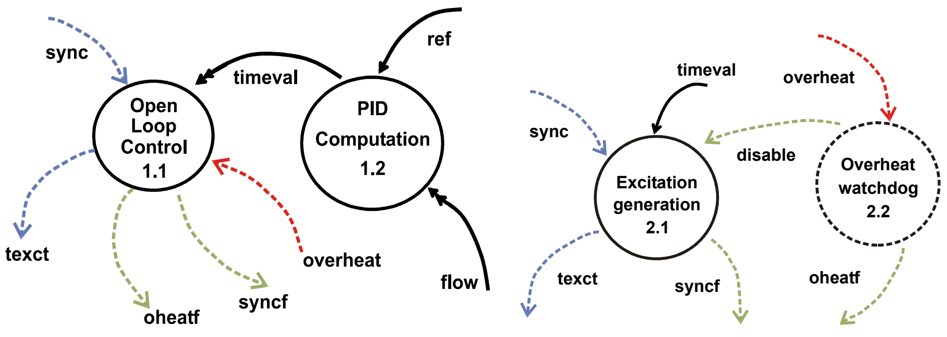
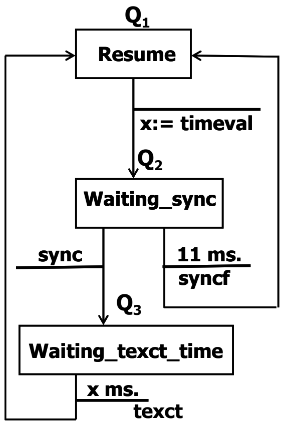


Fig. 3. (a) The closed loop control model. (b) The open loop control model

3.2 The Open Loop Control Subsystem

The Open Loop Control process "explodes" into two new processes, a data transformation process, named *excitation generation* (*EG*), which generates the control flow *textc* needed to excite the *TriaC* at a given time, Fig. 3(b). The *EG* process actually carries out control, not data processing, but it must be represented in the flow diagram by a *DTP*, since, according to the *SA/RT* rules, it cannot be considered a *pure control transforming process* because it receives a continuous data flow (*timeval*) from the *PID* process. The continuous flow *timeval* must be modelled by a *SYNC* process. The *overheat watchdog* (*OWD*) monitors the temperature of the *TriaC* and disables the *TriaC* excitation signal if the temperature becomes too high.



Q_1= timeval ? x → Q_2

Q_2= [t, t+11].sync><t → Q_3

|]t+11, ∝].syncf → Q_1

Q_3=]t+x-0.1, t+x+0.1].texct → Q_1

DIS= disable → STOP

EG= (init><t → Q_1)||DIS

Fig. 4. State-Transition Diagram representing the OLC and the equivalent *CSP+T* process.

The *OWD* and *EG* processes are both primitive transformation schemes, which must be specified by a *state transition diagram* (*STD*) that can then be converted into a unique Stateflow chart. The *EG* control process cyclically repeats the states shown in Fig.4. When in the state represented by *Q_2,* if the *EG* process does not receive the synchronization signal within 11 ms, a *failure condition* must be signalled by raising the exception *syncf.* This period of time amounts to one half cycle of a 50 Hz input voltage function –i.e.,10 ms + 1 ms of error margin-. In the control state represented by *Q_3, EG* waits for x milliseconds before sending the next excitation signal to the *TriaC*; the maximum allowed error in the waiting interval before sending the signal is 0.1 milliseconds. In the case of the *OWD* process, a *STD* is defined with only one state to monitor the temperature of the *TriaC*. If the *overheat* event occurs, the exception *oheatf* is raised and the excitation signal must be disabled, which is denoted by the event *disable.*

3.3 Closed Loop Control Subsystem

By means of closed loop control (*CLC*) it is possible to maintain a constant angular speed of the rotor, which can be used to control the centrifuge velocity of a washing machine, the air flow through a filter, the speed of a vehicle, etc. As shown in Fig. 3(a), two processes are included in the *CLC*: Proportional-Integral-Derivative control (*PID*) and open loop control (*OLC*). The AC motor is regulated by the *PID* control through the speed error signal between the set speed (set-point) and the actual measured speed. Depending on the speed error signal value, the *PID* controller adjusts the *timeval* value to determine the next excitation time of the *TriaC*. The adjustment of *timeval* is achieved by increasing or decreasing its value, depending on whether the current rotor speed is over or under the reference speed, respectively. Exactly when *timeval* elapses the *OLC* process enables the signal *texct* that causes the changing in the motor speed. A static analysis of the system reveals that the shorter the *timeval,* the higher the speed that is achieved by the rotor with a non-linear relationship. If the *timeval* is outside the interval [0,1/freq*2], its value is saturated by the maximum or minimum value.

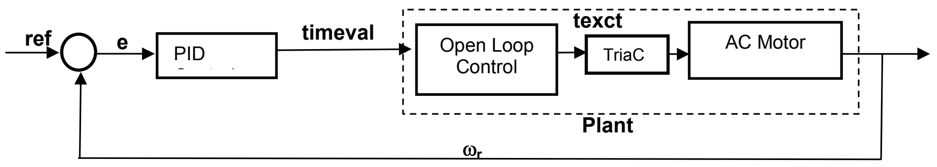


Fig. 5. Block diagram of a PID controlled constant air flow system

The *timeval* value must be specified as being a continuous data flow, rather than a discrete flow, in order to allow the two processes to act asynchronously. Two communications and one synchronization process –*SYNC*- are needed in order to implement an asynchronous communication in *CSP*.

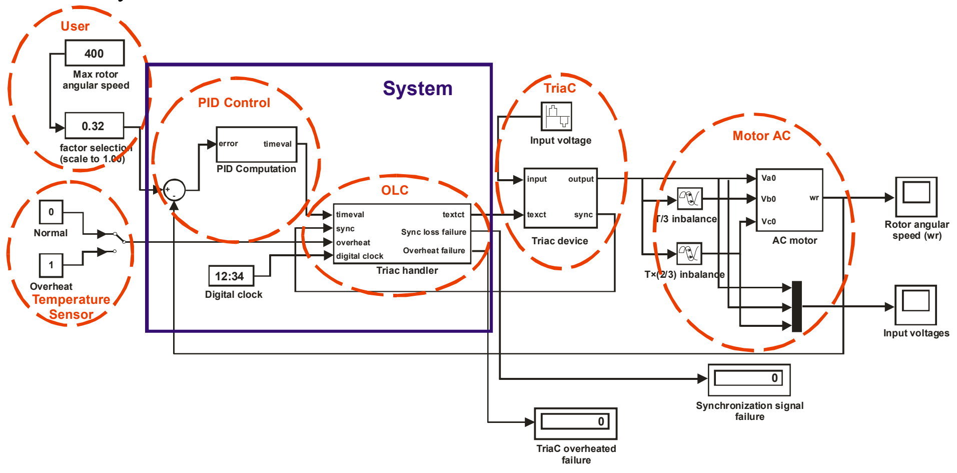


Fig. 6. Simulink model of the final system

4 Simulation of The Hybrid System

To obtain a simulation of the hybrid system proposed in this paper, more components must be added to the model. As shown in Fig. 5, we need to construct a model of the *TriaC* device, the AC motor, the flow sensor (or another sensor to measure the rotor speed), the temperature sensor and the sync generator. This model can be implemented in the Simulink/Stateflow framework as shown in Fig. 6. The most difficult component to model is the AC motor.

4.1 Physical Modelling of an Induction Motor

The functioning of an induction motor is based on the principle of mutual induction between electrical circuits traversed by a variable magnetic flux Φ. According to

Faraday's law, which is given by the following equation, $\varepsilon=-d/dt\,(N\cdot\Phi_B)$, the magnetic flux traversing a motor winding only depends on the current conducted by the circuit. A *self-induction* constant L can be assigned to any circuit affected by magnetic induction, according to the equation $N\cdot\Phi_B = L\cdot i_{reel}$.

$$\frac{dF_{ds}}{dt} = \omega_b\left[v_{ds} + \frac{\omega_e}{\omega_b}F_{qs} + \frac{R_s}{\chi_{ls}}\left(\frac{\chi^*_{ml}}{\chi_{lr}}F_{dr} + \left(\frac{\chi^*_{ml}}{\chi_{ls}} - 1\right)F_{ds}\right)\right]$$

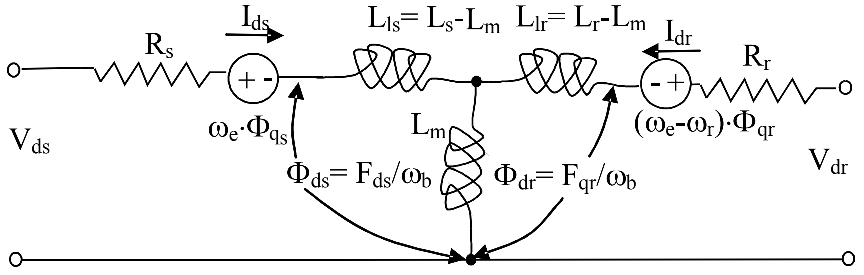


Fig. 7. Dynamic equivalent circuit of the stator linkage F_{ds}

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

d: direct axis of the rotating reference system	$\chi^*_{lm}= 1/(1/\chi_{ls}+1/\chi_{lr}+1/\chi_m)$: total reactance with the loses for magnetizing (χ_m)
q: quadrature axis	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
$F_{ij}=\Phi_{ij}$, where i=q or d and j=s or r, magnetic linkage	J: inertia momentum
v_{qs}, v_{ds}: stator voltage	M_e: motor electrical torque (output variable)
v_{qr}, v_{dr}: rotor voltage	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 frequency of the motor voltage.
χ_{lr}: rotor reactance $(\omega_e L_{lr})$	ω_r: rotor angular speed (output variable)
d: direct axis of the rotating reference system	$\chi^*_{lm}= 1/(1/\chi_{ls}+1/\chi_{lr}+1/\chi_m)$: total reactance with the loses for magnetizing (χ_m)

Nowadays, the winding of induction motors is carried out by three windings, each one bearing a voltage phase separated by 2π/3 rad. from the next one, which yields a rotating magnetic field in the stator. The velocity of rotation is called the *synchronous speed* w_e. Only if there is a small difference between the two rotation velocities w_e and w_r will an electrical torque be produced by the motor. This difference, named *slip*, is also given as a parameter of induction motors.

We can assume a reference system that rotates at the synchronous speed ω_e to ease the representation of the rotating magnetic field $\vec{B}$ and inductance linkages by a system of coupled differential equations that describe the physical induction and motor dynamics. This rotating reference system is known in the literature [12] as *dq* reference system. The induction motor is therefore modeled by two reels, the first one, situated on the *d-axis –direct-,* is aimed at conducting the current in the stator and it also generates the rotating magnetic field. The second one, on the *q-axis –quadrature-,* generates the induced magnetic field in the rotor. The differential equations that represent the magnetic linkage F_{ij} ($=\Phi_{ij}\cdot\omega_e$) between the stator and the rotor windings can be derived by applying fundamental electromagnetic laws to the equivalent electrical circuits. For instance, according to the physical model of the induction motor, the variation of the *d*-component of the stator magnetic linkage, F_{ds}, can be expressed in terms of the other linkage components that have an effect on it, as shown in Fig.7. The other components, 1 of these for the stator (F_{qs}) and another 2 for the rotor (F_{dr}, F_{qr}) expressed according to the *dq* reference system, are similarly obtained. Thus, there are five differential equations: 4 of them describing the change of the magnetic linkages and the fifth connecting the synchronous acceleration of the rotor with the electrical torque T_l generated by the motor.

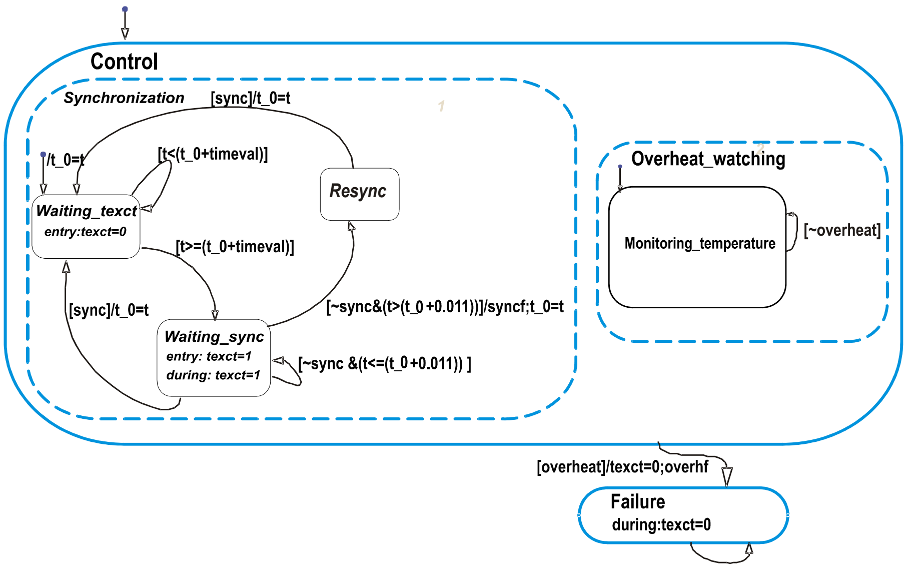


Fig. 8. Transformation of the OLC specification into a Stateflow Diagram

This model allows us to describe the magnetic coupling between the stator and rotor windings of an induction motor very accurately. The constants and system variables of the differential equations describing the induction motor are given in table I

4.2 Stateflow Model of the *TriaC* Controller

The imprecision regarding time specification in the *RSM* of any process has been overcome by deploying CSP+T process terms as a meta-notation to assign time constraints to the state-transition diagrams (*STD*) of compulsory use in *SA/RT*. The transformation of *STDs* to process terms is carried out by a system of rules. Then, the information required to specify the dynamic behaviour of a system of processes representing, for instance, the handler of a *TriaC* device, as in the example being discussed here, is systematically adapted to a functioning *Stateflow* diagram, as shown in Fig.8. This diagram fully describes the *logic* needed to implement the intended controller. The execution of the *Stateflow* diagram simulates the behaviour of the *CSP+T* process specification in Fig.4.

4.3 Matlab /Simulink Model of an Induction Motor Drive

The model of the induction motor `http://lsi.ugr.es/~sc/investiga/motor`, similarly to the one in [12], is structured in 3 main blocks: (1) transforms the three stator voltages: v_a, v_b, v_c , with a phase of 2π/3 between each two, into the rotating reference system *dq* (*direct-quadrature-axes*): v_{qs}, v_{ds}; (2) the block representing the *induction motor* itself (which inputs the three phase voltages, the synchronous angular speed ω_e and the load torque T_l), yields the three phase stator currents, the electrical torque T_e and the rotor velocity ω_r; (3) this block returns the expression of the model variables in the *dq* reference system to the *abc* (*three-phases*) reference system, since the latter gives us the standard graphical representation of currents in the stator. Two specific blocks have been designed to calculate the electrical torque M_e given by the motor and another to calculate the rotor axis angular speed ω_r. All the physical model constants given in table 1 have been defined using IS physical units in an *m-file*.

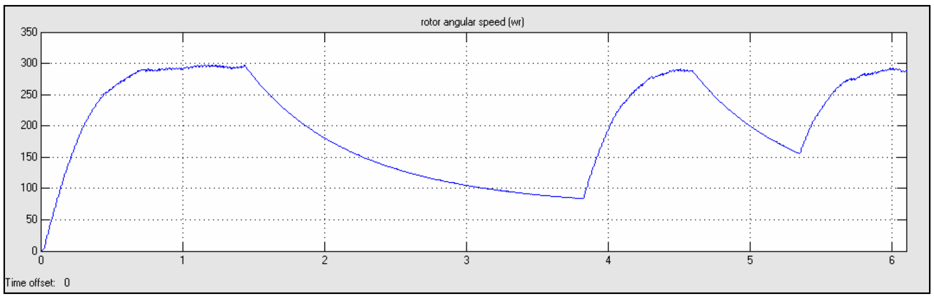


Fig. 9. Response of the rotor to changes in the stator speed

4.4 Results Obtained

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 fixed interval of time was the *TriaC* excited in every cycle. In this case the disturbances in the system response (ω_r) are remarkable, as shown in Fig.9. The 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 rapid oscillations around the new value for the rotor velocity.

By carrying out a simulation with the rotor velocity controlled by a PID, better results are obtained. Moreover, if we make a plot of the rotor angular speed output by the induction motor with respect to the reference speed, which is given by the input variable ω_e, we only obtain important oscillations at the beginning, while the system is trying to reach a stabilisation point. The oscillations shown in Fig. 10 only represent about 2% of the target value for the velocity ω_e, (320 rad/sec); these oscillations are caused by dynamic conditions during motor functioning, such as rotor axis friction.

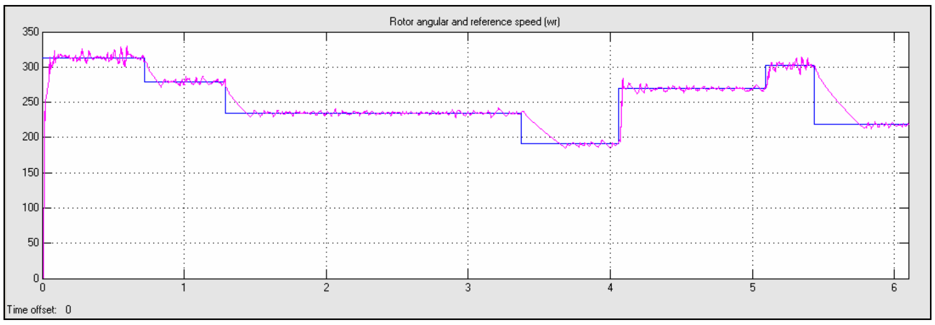


Fig. 10. Closed loop control model response to changes in the rotor speed

5 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 imprecisions 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 modelling *continuous and discrete dynamic systems,* have been used to integrate continuous components in a hybrid real/time system design. An application case is discussed in the article: the induction motor drive, and controlling an AC motor to maintain a constant air flow through a filter. The method is characterised by its easy integration within *ASE* environments and/or by formal tools based on *SA* notation. We are currently working on the development of a formal software tool based on *JCSP* [13] and Java, capable of automated specification, verification and code generation of real-time and embedded system software for several computing platforms.

Acknowledgments

The authors thank Professor Jose R. Balsas (Computing Engineering Department, University of Jaén) for his help with a first draft of this work.

References

1. DeMarco, T: System Analysis and Specification, Yourdon Press, 1971.
2. Borschev, A., Koleshov, Y., Senichenkov, Y.: Java Engine for UML based hybrid state machines. Proceedings of the 2000 Winter Simulation Conference, pp.1888-1894.
3. Cortés, L.A., Eles, P., Peng, Z.: Verification of Embedded Systems using a ri Net based Representation. 13th Int. Symp. System Synthesis(ISSS), 2000, pp.149-155
4. Capel, M.I., Holgado, J.A., Balsas, J.R.: A Transformational Approach to the Systematic Design of Real-time Systems. Manufacturing Engineering, vol 3, no. 2, pp. 5-13, 2004.
5. Ward, P.T., Mellor, S.: Structured Development of Real-Time Systems. Prentice-Hall, Englewood Cliffs (N.J.), 1985.
6. Hatley, D.J., Pirbhai, I.A.: Strategies for Real-Time Systems Specification, Dorset House, New York, 1988.
7. Baresi, L., Pezzè, M.:Towards Formalising Structural Analysis. ACM Transactions on Software Engineering and Methodology, 7, 1998, 1, pp.80-107.
8. Fencott, P.C., et al.: 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, 1994, pp.681-702.ŽIC, J.J.: Time-Constrained Buffer Specifications in CSP+T and Timed CSP. ACM Transactions on Programming Languages and Systems, 16, 1994, 6, pp.1661-1674.
9. Hoare, C.AR.: Communicating Sequential Processes, Prentice-Hall, Englewood Cliffs (N.J.), 1985.
10. Žic, J.J.: Time-Constrained Buffer Specifications in CSP+T and Timed CSP. ACM Transactions on Programming Languages and Systems, 16, 1994, 6, pp.1661-1674.
11. Capel, M.I., Holgado, J.A.: A New Design Procedure for a Real-Time Continuous System Model. LSI Internal Report, 2004, http://lsi.ugr.es/~sc/investiga/reports, 16 pages.
12. Tang, L. Raman, M.F. A new direct torque control strategy for flux and torque ripple reduction for induction motor drive-a Matlab/Simulink model. IEEE International Electric Machines and Drives Conference, 2001, pp.884-890
13. Welch, P: Process Oriented Design for Java: Concurrency for All. In: Parallel and Distributed Processing Techniques and Applications, PDPTA 2001, Las Vegas, Nevada, USA, 2001.