MODELICA LIBRARY FOR LOGIC CONTROL SYSTEMS WRITTEN IN THE FBD LANGUAGE

A. Leva[1], F. Donida[2], M. Bonvini[3], L. Ravelli[3]
[1]Politecnico di Milano, Italy, [2]PhD student at the Politecnico di Milano, Italy,
[3]Graduate student at the Politecnico di Milano, Italy

Corresponding author: F. Donida, Politecnico di Milano, Dipartimento di Elettronica e Informazione
Piazza Leonardo da Vinci 32, 20133 Milano, Italy, donida@elet.polimi.it

Abstract. This paper presents an object-oriented model library, written in the Modelica language, for the simulation of logic control systems written in the FBD (Functional Block Diagram) language. The library is part of a larger research project, the aim of which is to allow the joint simulation of plants - modelled as systems of nonlinear, differential and algebraic equations - and of the relative control systems, written as per the IEC61131-3 industry standard. The presented FBD library contains interchangeable models of different complexity, so that the user can specify a control system either as a continuous-time or an event-based model. The former yields fast simulations, so the one can rapidly verify the correctness of a control *strategy* for a particular problem. The latter permits precise evaluations of the behaviour of a control *algorithm*, up possibly to achieve a full *replica* of the control system as implemented on a specific architecture. The possibility of moving transparently from one description to the other is of great help in control design, and is one of the major strength points of the presented library.

1 Introduction and motivation

For the purpose of control commissioning, or process/control co-design, an accurate representation of both the process and the control system is of paramount importance ([3, 11, 15, 6, 17, 4]). However, if at an abstract level both can be represented as systems of equations, when it is to detail the description, the stories of the process and the control system definitely diverge: while the former is inherently described as a dynamic system, typically in the continuous time, the latter sooner or later needs turning into an algorithm ([7, 15, 16]).

From the standpoint of the analyst, two situations can be easily distinguished. In the first one, the problem is to structure a control strategy, assess its suitability to the plant and the requirements, and possibly parametrise the regulators. For that purpose, any peculiarity of the host architecture and the control code realisation *must* be irrelevant. In the second one, the problem is to assess the correct *implementation* of a strategy, i.e., to determine whether or not the realised algorithms match the controller models ([8, 18, 15]).

In addition, if one aims at a complete analysis, there are some functionalities of regulators that can only be described as algorithms (or, better, that would be too complex and unnatural to cast in the framework of dynamic systems): an example is given by some types of antiwindup mechanisms in PI/PID controllers. Fortunately such functionalities do not affect significantly the structuring of a control strategy, but in the parametrisation phase accounting for them may be necessary at quite early a stage.

In synthesis, one would like to choose the level of detail for the representation of the control system in a transparent manner, i.e., specifying that system adhering to some standard, and then simply selecting which type of representation (continuous-time model or algorithm) to use ([2, 9, 5, 10, 12]).

Nowadays, more and more control systems are implemented adhering to the IEC61131-3 standard, that defines five programming languages (Ladder Diagram or LD, Sequential Functional Chart or SFC, Functional Block Diagram or FBD, Structured Text or ST, Instruction List or LD) basically oriented to *logic* control, although most systems adhering to the standard also offer *modulating* control functions. In the last years, the IEC61131-3 standard has become very popular for example - but not only - in the arena of PLC (Programmable Logic Controller) programming, therefore spreading out in a vast number of contexts and applications ([18, 17, 6, 8, 3]).

In parallel, the object-oriented modelling (OOM) paradigm has been encountering in the last years a steadily growing success. With reference to the scope of this study, the main advantages of OOM are the following ([16, 1]).

- Models can be written in a declarative, a-causal form, via the abstraction of the so-called *connectors*, so as to be totally independent of the boundary conditions. For example, the model of a resistor is merely composed of the equation $V - RI = 0$, where V, R and I are respectively voltage, resistance, and current. The possibility of writing a model without having to care about the way it will be connected to others is of great help for the analyst to concentrate on relevant facts only, and fosters an effective model reuse.
- It is possible to mix a-causal models, based on declarative *equations*, and procedural models, based on *algorithms*. This means that, when modelling a controlled plant, the control system can seamlessly be

described with dynamic systems, oriented or not, with a *replica* representation of the control *code*, or any combination thereof. The implications of that possibility for simulation studies involving control synthesis and/or commissioning are dramatic.

The research presented herein is part of a larger project, the final goal of which is to exploit the OOM paradigm to create a modelling and simulation framework where one can:

- describe the physical process to be controlled with a system of Differential and Algebraic Equations (DAE) of virtually arbitrary complexity, maximising modularity and reusability thanks to the connector abstraction, and freely including any combination of first-principle, black- and grey-box models, depending on the peculiarities of the problem at hand, having also the possibility to describe the same (part of the) process with models of different complexity, based on the particular needs of the simulation study's
- describe the control system with any combination of continuous-time and event-based models (more precise equivalents of equation- and algorithm-based models in the OOM terminology), to allow structuring a control strategy in the idealised framework of continuous-time dynamic systems, and then having the simulation environment follow the implementation of that strategy down to the verification of the correct behaviour of the control code implemented in the chosen architecture.

In particular, the contribution of this manuscript (and therefore its organisation) can be summarised as follows.

- A discussion is carried out on the importance of describing control systems in a way compatible with industry standards. The key points of the problem are evidenced, and a possible framework to address the matter is identified, where OOM plays a relevant role. A couple of simple examples is reported to back up the previous statements.
- The above framework is applied, by presenting a free (GPL) library in the Modelica language, developed at the Politecnico di Milano, and covering (at present) the FBD language.
- A simple case study is briefly described, to show that with the proposed approach situations of realistic complexity can be addressed.

Having the IEC61131-3 standard available in a modelling and simulation environment is of great help, for at least two reasons. First, if an industry standard is uniformly adopted, there is (ideally) no room for ambiguities in the communication between the people who own and/or run the plant, and the analysts who create simulation models and realise with them the necessary studies; in fact some issues may still arise owing to the fact that virtually every standard is the result of a compromise, and therefore very frequently exists also in the form of so-called "dialects", but addressing such problems is apparently beyond the scope of this research. Second, the solutions found at the simulation level are deployed to the target control architecture in a very straightforward way.

2 Modelling and simulating *controlled* systems

2.1 Foreword

It is obvious that any simulator has to be as close as possible to the simulated object. It is also obvious that such a general statement needs declining into the particular needs of the study for which the simulator is built. But in the particular case considered, further reasoning is in order from (at least) two points of view, namely (a) the necessity to have the simulator follow the overall design path in the most effective way, and (b) the inherently heterogeneous nature of the object to be simulated ([13, 7]).

From the first point of view, the design of a complex (controlled) system is typically a top-down process. One starts with a general picture of the system, then details some part of it, then possibly goes back to a less particular level, then down again into some detail, and so back and forth till the design is complete ([8]).

At every step of the process, there are needs to fulfil, thus decisions to take, and constraints to be obeyed, that frequently include the effects of decisions previously taken. Quite often simulation is the way of election to obtain the information needed to take the next decisions, but for such a support to be effective, it must be possible to include in the simulation all the information available, and it must not be necessary to guess about any decision to be taken subsequently. If all the plant (let alone the control) has to be fully specified before running the first simulation, there is little use in simulating. And the same is true if it is not very easy to reconsider previous decisions, take different choices, re-simulate, and repeat the above cycle as many times as needed.

On a similar front, when simulating to answer a certain question, most likely it is not required to represent the whole system at the maximum level of detail possible with the available information. Almost invariantly the question to be answered concerns a part of the system, and to obtain the necessary data the rest of the system can be safely replaced with correct boundary conditions for the part on which the simulation concentrates. It is quite obvious that to do so effectively, and in particular within the decision cycle just mentioned, for each part of the

overall system models with different levels of complexity are needed, and those models must be interchangeable in a transparent manner.

From the second point of view, the *outcome* of the design of a controlled system is composed of two very different parts: the "plant" is invariantly a physical object, the "control" is most frequently (always, in the context of this work) a computer architecture running one or more programs that implement control algorithms ([2, 3, 14, 18]).

Also in this case it is advisable to have interchangeable models of different complexity, for the same reasons as above, but there is an additional important necessity. In some phases of the design it is correct to describe the two with the same formalism, typically that of continuous-time dynamic systems. This is for example the case when one is structuring a control strategy at quite general a level, and wants to know whether or not that control structure is suitable for the problem at hand, irrespectively of how the corresponding algorithms and code will be obtained. In those phases, the designer wants to reason in terms of dynamic systems (for example, a PID controller is at this stage a transfer function). The overall (controlled plant) model is then a DAE system, the evolution of which is time-driven. For both historical and practical reasons, one tends here to think of *continuous* time.

In other phases it is necessary to represent the detailed behaviour of the control code, typically when it is necessary to decide whether or not a control strategy, previously assessed and considered to be correct, is implemented in the right way from the algorithmic standpoint, or when one has to investigate the possible effects of the particular architecture, including typically real-time and network communication management, on the behaviour of a certain control algorithm implementing a certain control strategy. In such phases, the plant model is still a DAE system, while the control (model) is a set of programs that evolve driven by *events*, typically generated by one or more clocks that dictate the sampling and actuation process, and synchronise the computation of control signals by means of convenient algorithms. Most functionalities of industrial regulators (for example, antiwindup or bumpless auto(manual switching) are better specified as algorithms than as dynamic systems, and that is why this is the stage when those functionalities typically come into play in simulations aimed at control synthesis. Therefore, at this stage the PID above is typically a program that computes the next value of the control signal, the execution of which is triggered by some clock.

For the reasons illustrated above, it is necessary that the designer can freely interchange controller models of the two types that consistently with the core of the problem (and the OOM terminology, by the way) are from now on termed respectively "continuous-time" and "event-based". Mixing time- and event-driven evolution is in general a very complex problem, but in the particular case of IEC-compliant control systems, some considerations allow to introduce a straightforward simplification, that is exploited by the modelling approach adopted in the presented library. The key point, discussed in the following section, is how events need to be managed in the IEC framework.

2.2 Modelling IEC 61131-3 controls

The IEC 61131-3 standard defines five programming languages: three are textual, two graphic. The modelling approach used here can be exemplified by starting from the implemented language, namely the FBD. The key point in the discussion is that a controlled plant can be modelled, in total adherence to the way IEC-compliant controllers are realised, by greatly simplifying the problem of managing events.

In the IEC 61131-3 framework, CPUs operate according to the so-called "massive copy" cycle. The CPU clock periodically triggers a cycle, which is composed of three phases:

1. inputs from the process are read and stored to memory (and therefore, from the CPU's point of view, do not vary till the beginning of the next cycle),
2. the status of the control program is updated by executing suitable procedures that compute the new state of each block, hence obtaining the control outputs, that are written to memory,
3. and finally the control outputs are transferred to the physical outputs, which therefore vary only at the end of each cycle.

In such a framework, the only possible sources of events are the CPU clock(s), and the occurrence times of events is deterministically known. Of course this mean neglecting possible timing imprecision or glitches, which however are beyond the scope of the studies that can be done with the presented library: the goal here is to determine what happens on a given architecture when it functions correctly, not what could happen if its timing mechanism fails. As a result, the simulation overhead introduced by the implementation of control blocks as event-based systems in the particular case treated here is definitely acceptable.

Also, the same case suggests a possible, very useful improvement to Modelica translators. At each control step (event), the new control signals (inputs to the DAE system representing the process) depend either on the present value of process outputs, or on quantities relative to the control system that however were already known when the previous control signals were computed. It would therefore be possible to detect whether or not any of the process DAE inputs changes at the event, and if none changes, simply proceed in the integration of the process DAE just as if no event occurred. With a suggestive (yet maybe not completely rigorous) terminology, one could think of

introducing the idea of "tentative event" as something that *could* force the integration of the process DAE to stop, but does so only if it is revealed that triggering the event actually causes some process input variation.

3 Library organisation

The library comes in a single Modelica package named `FBD`, and organised in subpackages.

- the `FBD.OneBitOperation` subpackage implements basic logical operations,
- the `FBD.CompareOperation` subpackage implements comparisons (the <, >, >= ,<= ,= operators) on the Integer and Real types,
- the `FBD.Counter` subpackage provides up/down counters,
- the `FBD.MathOperation` subpackage implements the basic mathematical instructions,
- the `FBD.Timer` subpackage provides timers (and is similar to the Counter one),
- the `FBD.NBitOperation` subpackage implements logical operation on arrays of bits,
- the `FBD.LinearSystems` subpackage provides linear, time invariant dynamic systems in the continuous and discrete time, as typically specified in IEC-compliant control code development environments,
- the `FBD.IndustrialController` subpackage contains several industrial controllers, including of course several types of PID,
- the `FBD.Test` subpackage contains test simulators for each FBD block, individually, to allow for a precise comprehension of its functionalities,
- and finally the `FBD.Applications` subpackage provides some examples of use of the FBD blocks of the library.

For each component a "test" model is provided, to allow the user to fully understand how that component works, and possibly disambiguate situations where the available specifications are not fully univocal.

Also, especially for regulators and in accordance with the considerations above, both continuous-time and event-based models are present: the former type of model, as said above, for fast simulations to check strategies, the latter for detailed (slower) simulations to check algorithms. The library therefore allows to perform both types of simulation, and even to mix the two, e.g. by convenient use of model replaceability, and top-level variables. To limit the performance loss, equations (not algorithms in the Modelica sense) were used in event-based models, so as to allow those models to be manipulated with the rest of the simulator. Doing so involves some limitations when porting a pre-existing algorithm into the library, since for example multiple assignments are not allowed. It is the authors' opinion, however, that an accurate translation in the form adopted by the presented library is possible for of any control algorithm of practical interest.

To show the completeness of the library, figure 1 reports its structure as seen in the browser of the Modelica translator Dymola.

4 A simulation example

This section reports a simulation example. to show the usefulness of the possibility of simulating the same regulator as continuous-time and as event-based model. The example refers to some PI/PID control loops, and deals with set point step and ramp responses where the antiwindup mechanism of the regulator comes into play. The process to be controlled is described by the transfer function

$$P(s) = \frac{1}{1+2s+s^2/0.016} \tag{1}$$

and the PID regulator

$$R(s) = 10\left(1+\frac{1}{30s}+\frac{3s}{1+0.3s}\right) \tag{2}$$

is applied to it, in the continuous-time version and as an event-based model with a sampling time of 0.01 s.

Figure 2 shows the comparison between the continuous-time (R1) and event-based (R2) controller implementation in the case of a ramp response (left column of plots) and of a step response (right column): SP, PV and CS stand for Set Point, Process (controlled) Variable, and Control Signal, respectively. Apparently, simulating the same controller as a continuous-time or an event-based model (i.e., as it will really be implemented) can give very different results, depending not only on the controller parametrisation, the sampling time and other very well known facts, but also on the control law being incremental or positional, of the antiwindup type, and so on (facts that conversely are frequently overlooked). The example therefore backs up the usefulness of the presented library as far as the control behaviour evaluation is concerned.

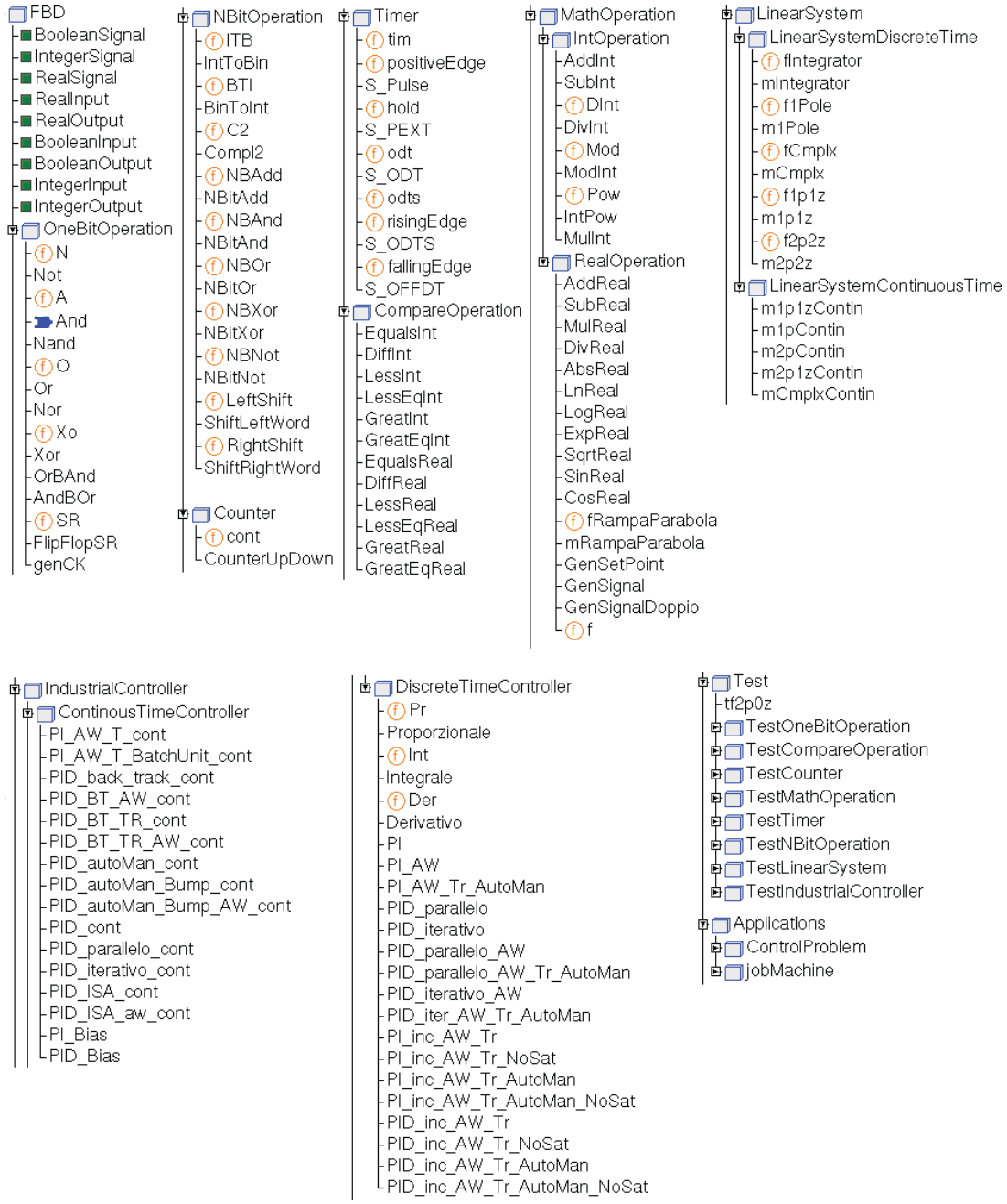

Figure 1: The FBD library as seen in the browser of the Modelica translator Dymola.

5 Conclusions

A free (GPL) Modelica library for the simulation of logic control systems written in the FBD (Functional Block Diagram) language was presented.

The library adheres to the FBD specifications as defined in the IEC61131.3 standard, and contains not only strictly logic blocks, but also the main types of industrial controllers, particularly of the PID type. The adoption of an industrial standard facilitates information sharing and greatly reduces ambiguities.

With the presented library, that the user can specify a control system as a continuous-time or an event-based model, for maximum flexibility in fulfilling the simulation needs.

Some simulations were presented to illustrate the usefulness of the library, which will be extended in the future, with respect to both FBD and other IEC-compliant languages.

6 References

[1] Modelica Association. Modelica. `http://www.modelica.org`.

[2] F. Auinger, R. Brennan, J. Christensen, J. L. M. Lastra, and V. Vyatkin. Requirements and solutions to

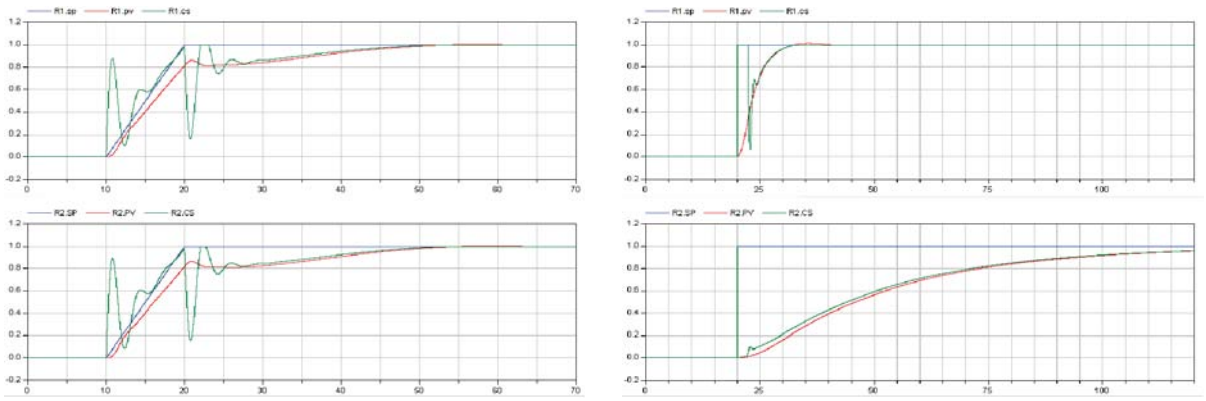

Figure 2: Results of the example where the control system composed of process (1) and regulator (1) is subject to a ramp- (left) or step-like (right) set point variation.

software encapsulation and engineering in next generation manufacturing systems: Oooneida approach. *International Journal of Copmuter Integrated Manufacturing*, 18(7):572–585, 2005.

[3] D. Barrit. Iec 61131 and dsdm in real-time process control applications. *Computing & Control Engineering Journal*, 2002.

[4] M. Bonfé, C. Fantuzzi, and L. Poretti. PLC object-oriented programming using IEC61131-3 norm languages: an application to manufacture machinery. In *Proc. IEEE/ASME International Conference on Advanced Intelligent Mechatronics*, volume 2, pages 787–792, 2001.

[5] DeltaV. DeltaV monitor and control software.

[6] F. J. Fraustro and E. Rutten. A synchronous model of iec 61131 plc languages in signal. *Real-Time Systems, 13th Euromicro Conference on, 2001.*, pages 135–142, 2001.

[7] L. Gheorghe G. Nicolescu, F. Bouchhima. Codis - a framework for continuous/discrete system cosimulation. In *2nd IFAC Conference on Analysis and Design of Hybrid Systems*, pages 274–275, Alghero, Italy, June 2006.

[8] G. Hassapis. Soft-testing of industrial control systems programmed in iec 1131-3 languages. *ISA Transactions*, 39:345–355, 2000.

[9] ISaGRAF. Iec 61499 execution model.

[10] ISaGRAF. ISaGRAF home page. `http://www.isagraf.org`.

[11] Y. Itoh, I. Miyazawa, and T. Sekiguchi. Modeling of a sequential control system with cyclic scan by petri net. *Electrical Engineering in Japan*, 139(4):461–467, 2002.

[12] O. Johansson, A. Pop, and P. Fritzson. Engineering design tool standards and interfacing possibilities to Modelica simulation tools. In *Proc. 5th International Modelica Conference*, 2006.

[13] E. Kofman. Discrete event simulation of hybrid systems. *SIAM Journal of Scientific Computation*, 25(25):1171–1797, 2004.

[14] T. Li and Y. Fujimoto. Control system with high-speed and real-time communication links. *IEEE: transaction on Industrial Electronics*, 55(4):1548–1557, 2008.

[15] P. J. Mosterman, G. Biswas, and J. Sztipanovits. A hybrid modeling and verification paradigm for embedded control systems. *Control Engineering Practice*, 6:511–521, 1998.

[16] M. Shirakawa. Development of a thermal power plant simulation tool based on object orientation. volume 220, pages 569–579, 2006.

[17] J. R. Silva, I. Benitez, L. Villafruela, O. Gomis, and A. Sudrià. Modeling extended petri nets compatible with ghenesys iec61131 for industrial automation. *International Journal of Advanced Manufacturing Technology*, 36:1180–1190, 2008.

[18] H. A. Thompson, D. N. Ramos-Hermandez, J. Fu, L. Jiang, I. Choi, K. Cartledge, J. Fortune, and A. Brown. A flexible environment for rapid prototyping and analysis of distributed real-time safety-critical systems. *Control Engineering Practice*, 15:77–94, 2006.