Extended Generic Layered Architecture for Real-Time Modeling and Simulation

Jacek Malec[1] and Janusz Zalewski[2]

[1]Dept. of Computer Science, Lund University
S-221 00 Lund, Sweden
jacek@cs.lth.se
[2]Computer Science Program, Florida Gulf Coast University
Ft. Myers, FL 33965, USA
zalewski@fgcu.edu

Abstract

This Generic Layer Architecture (GLA) has been created to deal with problems of reasoning in conventional process control. It is composed of a process layer, dealing with the external environment, a rule layer, that provides an inference engine associated with a system state, and an analysis layer responsible for planning and reasoning. This results in a hybrid controller that enhances conventional modes of operation by the capability of taking intelligent decisions. In this work, we are extending the concept of GLA by adding into the picture three new environments, to make it suitable for purposes of modeling and simulation. The broader context includes a user interface, network communication and image database, where GLA becomes a core element of a sophisticated real-time information processing system. The extended architecture is applied to a simulation problem from air traffic control - reasoning about, predicting and detecting collisions. In this problem we focus on investigation of timing estimates on the reasoning process to meet hard deadlines in the problem domain.

Introduction

By the term *embedded systems* one usually refers to computer programs embedded in physical systems. They provide control to mechanical, chemical and other kinds of plants. The design and implementation of embedded systems usually involve the contribution from both control and computer engineers (Sanz and Zalewski, 2003).

Simple embedded systems might be designed and implemented using rather straightforward approach. The dynamics of the system in hand needs to be studied and a control algorithm may be inferred possibly using some appropriate tool, such as differential equations or a Laplace transform. An implementation of such algorithm amounts to choosing an appropriate execution model (usually periodic computations) and a computer platform, and then generating code for that platform that provably (this issue is unfortunately often omitted) implements the algorithm.

As systems become more and more complex, a plant cannot be any longer described as a set of (linear) differential equations. Firstly, such a description would take too much time to create; secondly, it would be of no use (due to complexity) for the purpose of creating a control algorithm. In such cases techniques like hierarchical description and modularization come to use.

One of the modern ways to approach the problem is to use the hybrid system models: a system is described by a family of *modes*, where each mode is described by a set of differential equations (Grossman et al., 1993). The supervisory control consists of switching among modes, depending on the state of the plant.

Such systems are called *hybrid*, since they consist of continuous subsystems and a discrete supervisory structure. To implement hybrid controllers one needs to provide the algorithms for each control mode and the discrete controller to perform the mode switching and handle the interaction between the continuous and the discrete control.

Implementing hybrid controllers using languages like C or Ada is rather cumbersome since they offer very poor support for algorithms expressed using state machines. Software support and computer aided tools are therefore

necessary to assist in the design of hybrid controllers. One such tool is the Generic Layered Architecture, GLA, (Morin et al., 1992) for implementing hybrid controllers in software. The languages and tools developed for the architecture support both periodic and discrete computations required by hybrid controllers.

However, the software development tools are only a part of a larger picture - first the software needs to be specified and designed. Even better if this specification could be formal, so that validation and verification techniques could be used with their full potential, yielding provably correct embedded system. But, in order to get confidence in formal tools, one needs a faithful model of a (hybrid) controller and its environment.

The GLA has been previously used as a modeling tool (Malec et al., 1995), however at that time no verification support tools have been incorporated in the GLA methodology. Due to increasing complexity of considered applications, but also due to the availability of the more powerful computer technology, the next step towards a more powerful modeling and analysis methodology could be taken.

In this paper we describe the extended GLA (eGLA) obtained by adding into the picture three new environments, to make it more suitable for purposes of modeling and simulation. The broader context follows the fundamental principles of designing real-time software architectures (Zalewski, 2001) and includes a user interface, network communication and image database, where GLA becomes a core element of a sophisticated real-time information processing system. The extended architecture is applied to a simulation problem from air traffic control - reasoning about, predicting and detecting collisions. In this problem we focus on investigation of timing estimates on the reasoning process to meet hard deadlines in the problem domain.

The paper is divided as follows. First, the original GLA is introduced, together with the associated software and formal analysis tools. Then the extensions to the architecture are described. Finally, we present the problem from air-traffic control and sketch the advantages of using eGLA in this context.

The Generic Layered Architecture

Our approach to hybrid system design and implementation builds on the three-layered software architecture GLA, the generic layered software architecture, developed since the early nineties (Morin et al., 1992). The architecture differs from other layered approaches by grouping similar types of computations into *layers* (shown in Figure 1), as opposed to functional decomposition.

The first layer, called the *process layer* (PL), is intended to host implementations of numerical, periodic tasks, such as identification or control. Data handled by this layer are contained in input and output vectors. Computations have the form of mappings from input vectors to output vectors and are performed periodically in synchronization with the sample rates of sensors.

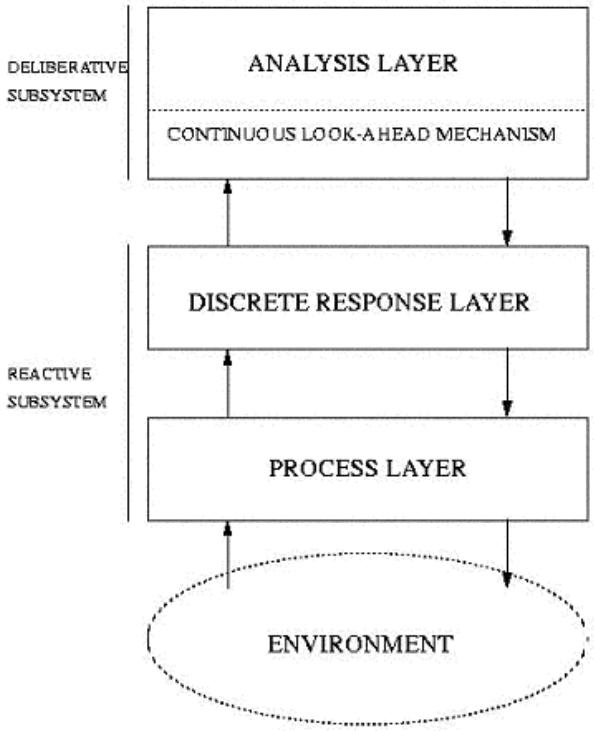

Fig. 1. The generic layered software architecture.

The middle layer is called the *discrete response layer* (DRL), and performs tasks, which are by nature asynchronous. For instance, it computes the response to asynchronous events that are recognized in the PL. An example of such a task is the change of control mode (of the PL) due to the change of mode in the environment. The computational model assumed for the DRL is that of discrete event systems (DES). There exist several equivalent DES formalisms: automata, transition systems, rule-based systems, etc. All of them distinguish the notions of *state* and *transition* as central, although the details vary from model to model. We have used the rule-based approach for the purpose of specifying knowledge-based system prototypes. However, this does not preclude usage of other approaches (Malec, 1992).

The top layer is called the *analysis layer*, because it is intended to handle symbolic reasoning tasks such as prediction, planning and scheduling, which require reference to physical time. The output of this layer can be either control events that guide the DRL in its decisions or

parameter settings that are passed through the DRL and directly affect the operation of the PL.

Real-Time Software Tools

This architecture and its implications on software engineering issues have been thoroughly studied in previous research (Morin et al., 1992). One of the conclusions was that it facilitates prototyping of systems, especially because it allows development of generic software tools, which can be used for implementation of each particular application system. Along this line software kernels, or *engines*, have been developed for construction and implementation of the process layer and the discrete response layer. The set of tools includes:

- Process Layer executive, PLX (Morin, 1993): a multi-threaded time-triggered real-time engine for implementation of process layer software. It has been implemented on a pSOS based system, on a PC running VDX, and recently on a RTLinux-based PC. There also exists a simulator of the PLX, which runs under Unix;
- Process Layer Configuration Language, PLCL, (Morin, 1991) and its compiler: a language for specification of PL module interconnections and interfaces to both sensors and actuators on one side, and to the DRL software on the other side. The modules themselves are programmed in a subset of some conventional language, such as C or C++;
- Rule Layer executive, RLX (Morin, 1994): an engine for implementation of rule-based discrete-event systems in the DRL. It has been implemented on Unix-based machines, and on a PC with VDX and RTLinux;
- Rule Language. RL (Morin, 1994): a rule-based language for declarative specification of discrete-event control.

The Process Layer Executive

The PLX supports the implementation and maintenance of the hard real-time parts of the application, which perform the transformations of periodic data. During processing, all data are stored in a *dual state vector*, which is a global data structure consisting of an input and an output vector. The values in the input vector represent either sensor readings or internal state, whereas the values in the output vector represent actuator outputs or new internal state.

A PL application defines a sequence of transformations that should be applied to the input vector in order to compute a new output vector. Since sensor values are read periodically, the transformations have to be applied with the same periodicity. When the transformations have been applied, actuator outputs are flushed to devices and the new internal state is installed in the input vector, thus establishing the new state of the process layer.

The PLX engine supervises a PL application, which includes managing the internal state, reading sensors and writing to actuators using user defined access functions, and supervise the execution of the periodical transformations of the dual state vector. The PLX supports decomposition of the vector into several sub-vectors, each of which has its own period, causing transformations to be executed at different rates.

The PLX engine and the PLCL compiler together form the basis for a worst-case execution timing analysis of a PL application (Morin, 1993).

The Rule Layer Executive

The rule layer executive supports the implementation of rule-based event processing. In the RLX the state of the world is represented by the time dependent values of a set of symbolic state variables called *slots*. A slot is updated only when its value changes, due to an external event or as a result of a change of another slot's value. Rules specify dependencies between slots, and typically have the following form: *if the value of a particular slot is changed in a certain way, then the value of another slot should also be changed as a result*. Internally, each such change may trigger additional rules, which lead to more updates. This forward chaining process may continue in several steps.

We take an object-oriented view of the rule base in the sense that we associate a set of rules with each slot. This view facilitates the flexibility and maintainability needed for complex systems.

The RLX tool has two major tasks. Firstly, it maintains the set of slots and rules, which determines the behavior of the discrete response layer. Secondly, it performs the forward chaining of rules. The forward chaining is typically initiated by an event recognized by the PL. The result of the forward chaining process may be the change of control algorithm used by some PL process or direct output to a device interfaced by the PL.

The RL is essentially a syntactic variant of a simple temporal logic and is used for declarative specification of behavior of the discrete response layer. A comprehensive set of tools for correctness and consistency checking, for timing analysis of RL programs and for code generation have been developed during the recent years (Lin and Malec, 1998; Lin, 1999a; Lin, 1999b).

The Extended GLA

The GLA serves well as a medium for embedded controller implementation, but its use as a tool for system modeling and analysis, although foreseen from the very beginning (Malec et al., 1995), is somehow limited. This is mainly due to the lack of support tools that would help the

engineer to formulate the problem in a language of their choice, create a simulation for deeper understanding and finally perform analysis using formal and informal tools available. In addition to that, the model of the controller can be extended to include not only the process aspects currently implemented in the Process Layer, but also three additional aspects of an architecture a of modern real-time controller, as illustrated in Figure 2 (Zalewski, 2001):

- Graphical User Interface, GUI, allowing the operator to interact in real time with the controlled process and other components of the control system;
- Network Communication, handling the connectivity among multiple processors or computing nodes forming the control system; and
- Database Interface, allowing for real-time access to permanently stored data, such as images in real-time simulation, flight plans in air-traffic control, etc.

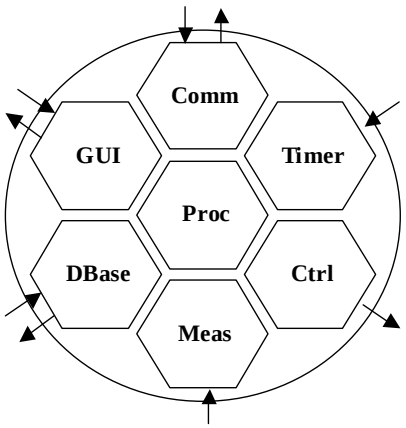

Fig. 2. Architectural template for real-time software design

Assuming that the GLA's Process Layer is represented in Figure 2 by the Measurement and Control components, and the upper GLA layers are embedded in the Processing component, the extensions proposed just add independent run-time agents, similar to the PL, concentrating on three orthogonal I/O functions. Therefore the extension should not be treated as additional layers of the architecture. The division of GLA into three layers, based on the computational pattern employed, still holds.

However, some functionality should be available to the end-users if the tool is to fulfil its functions. Thus the extensions should be seen as separate functionalities available at the lowest layer. All three extensions, User Interface, Network Connectivity, and Image Database, are described briefly below.

User Interface

No software, whether it's the controller itself or the tool to design it, can expect popularity without an intuitive, easy-to-use GUI. In an early phase of language design, there are usually no additional tools, but before such development can be considered mature there needs to be a number of user interfaces allowing a non-expert user to exploit all the advantages of the tool without making unnecessary mistakes.

GLA was lacking the user interface for years, until very recently, when the design and implementation have begun. The currently developed interface allows the user to develop PL programs and test their timing properties, to develop RL programs and run various semantic and timing checks, to develop lookahead reasoning in the analysis layer and test its connection with the lower layers of the software. There exist also tools allowing import of specifications done using external formalisms, like e.g., the hybrid automata developed using HyTech tool. Moreover, a stepping and debugging facility exist for all three layers of the software.

At the tool level, we consider introduction of visualisation mechanisms for better illustrating the interplay between the discrete and the continuous part of the software, between the system and its environment (including other embedded software developed using the GLA approach, see below) and of the analysis layer software.

Network Connectivity

In order to use a GLA-based system in a larger setting, including other computer-based control or, especially, simulation systems, one needs to provide connectivity to the outer world. The natural choice for this is TCP/IP networking allowing pieces of software running in different systems to communicate with each other according to predefined protocols.

In general, all three levels of the GLA architecture can communicate smoothly over the network with other networked applications, possibly but not necessarily also GLA-based. However, in this project we are restricting the network connectivity to the lowest layer agent responsible exclusively for maintaining data communication. This allows distributed algorithms, such as large-scale simulations, to be easily developed and run. The applications described below provide a good example of advantages of distributed simulation and power of analysis tools available for the GLA.

Image Database

Current state-of-the-art simulation and analysis tools require access to large databases containing data in various formats. A particularly challenging topic is the use of large corpora of image data available. We expect that applications developed using GLA might exploit such databases and therefore the third extension of GLA is the mechanism of accessing them.

Usually such need arises at the level of discrete model, when a control mode is to be changed (e.g., classification of an observed plane as a civilian line aircraft rather than a military UAV, or classifying a ship as a ferry carrying people rather than a cargo ship), but the implementation of the mode change will usually rely on computations

performed on the lowest level. Therefore the main interface will be implemented at the process layer level rather than any of the other two layers.

Case Studies

Benchmarking and Training System Simulation

To verify the concept and evaluate timing properties of eGLA, several experiments were conducted within the experimental testbed composed of three high-level software design tools imitating the behavior of the Process Layer (VxWorks) and User Interface (ObjecTime and Rose Realtime), with SES/workbench acting as a Discrete Response Layer (Mathure at al., 2003). First, the tools were mapped onto a five-task benchmark as described in (Guo et al., 2003) and illustrated in Figure 3.

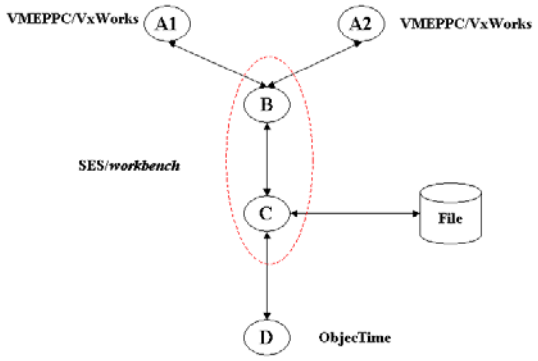

Fig. 3. Five-task benchmark simulation architecture.

The benchmark is consistent with the architectural template presented in Figure 2 and is composed of the following components:

- two measurement processes, A1 and A2, representing the Process Layer, run on VxWorks real-time kernel
- a computational process, B, representing the two upper layers of the GLA, run within SES/workbench
- a database process, C, representing a database interface, run also within SES/workbench, and
- a GUI process, D, run within ObjecTime.

The idea of this experiment was to evaluate performance of the model of an extended GLA, in a heterogeneous and complex environment, where multiple tools based on different theories (differential equations, finite state machine, and queuing theory), but integrated via TCP/IP protocol suite, contribute to the overall goal. The experiments proved that the integration is possible and stable operation is achieved within a short period of time.

The next step was to use the testbed for a more sophisticated case study consisting of a training system developed for combat vehicles (Al-Daraiseh et al., 2000). A general configuration of this system consists of the following four building blocks:

- SMC, Simulation Management and Control, which is a central part of the simulation, running within SES/workbench
- ITS, Intelligent Tutor and After Action Reporting, which runs under VxWorks
- IG, Image Generation Subsystem, also running under VxWorks
- CGF, Computer Generated Forces, simulated in ObjecTime/Rose Realtime.

The performance of this system was evaluated by simulating SMC module as a server in the role of the GLA's DRL layer. It served 9 clients delivering nine types of periodic messages and one additional client generating high-priority non-periodic traffic, according to the customer's specification (Al-Daraiseh et al., 2000). The impact of changing arrival rates on queue length was analyzed for both types of traffic. As shown in Fig. 4, sample measurements for periodic traffic only and three different distributions of the server population. The results show faster system saturation for longer service times, as expected (Mathure et al., 2003).

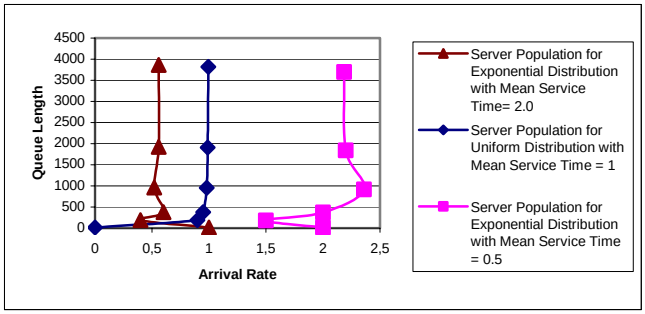

Fig. 4. Queue length vs. arrival rate for periodic and non-periodic traffic combined.

Air Traffic Conflict Detection and Resolution

The objective of this case study is to focus on the capability of the DRL layer in the extended environment and apply the Extended GLA to a novel algorithm suite for airspace management and deconfliction of multiple Unmanned Aerial Vehicles (UAV's). To avoid collisions in space, ideally all vehicles should have enough information of all other vehicles' positions, directions and speeds. This idea, however, is not feasible and effective to implement, since it is only good under the assumption that vehicles are cooperating and a computational unit in each vehicle has enough memory and processing power (speed) to complete computations on time, before a collision occurs. For example, for N vehicles, assuming each vehicle has to compute its X parameters for the next time period to predict collisions, the computational time is proportional to (N-1)*X, which is linear with respect to the number of vehicles, but has two disadvantages:

- grows significantly with a number of vehicles involved and may reach limits of processing capabilities, and
- is unpredictable in highly crowded environments, because there is no way of knowing in advance how big N can become.

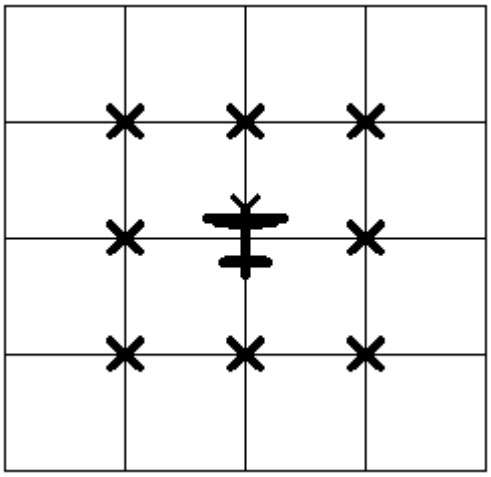

Fig. 5. UAV and its neighbor zones in 2D space.

Thus, a collision avoidance method is needed, which can offer reduction in computational space and time, additionally ensuring collision predictability under all circumstances. Initial reduction of computational complexity can be achieved by reducing the problem to a two-dimensional space and assuming that only immediate neighbors can participate in a collision, as illustrated in Figure 2. Then, the computational time reduces and is never greater than 8*X, which means that its upper limit does not depend on the number of vehicles involved.

A new method called Right-Of-Way (ROW) deconfliction further reduces the computational time to 3*X, because only three immediate right-hand neighbors are involved in computations, assuming that all vehicles in space observe the Right-Of-Way principle and use the same algorithm of deconfliction.

A block diagram of software architecture of ROW deconfliction is shown in Figure 6. The reasoning system consists of two major parts: Trajectory Prediction module and Conflict Detection module, both forming the DRL layer in eGLA. The Trajectory Prediction module takes inputs from its own vehicle and obtains global time. It also responds to inputs from Command and Control (equivalent of GUI). UAV inputs come from vehicle sensors. Based on this information the module estimates and predicts values of state of the vehicle, using Kalman filter as a computational model. The results of computations are periodically fed to the Collision Detection module, which combines it with information obtained from participating neighbor vehicles and with Command and Control to use it as a basis for decision making.

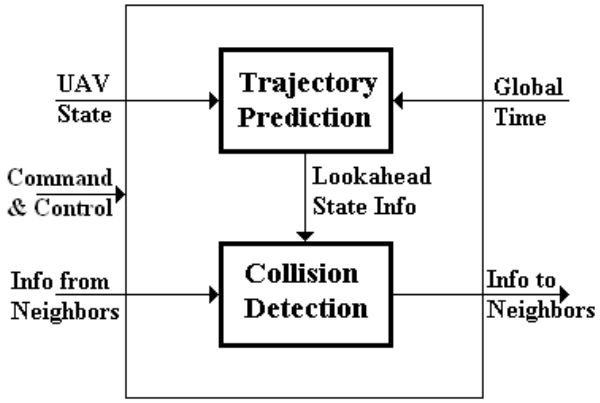

Fig. 6. Software architecture of the ROW system.

The model of the dynamics assumes a two-dimensional linear system described using a set of general state-space equations of the form (Johnson et al., 2003):

$$\mathbf{X}_{k+1} = \varphi_k \mathbf{X}_k + \mathbf{w}_k$$
$$\mathbf{Z}_k = \mathbf{H}_k \mathbf{X}_k + \mathbf{v}_k$$

where $\mathbf{X}_k$ is an (n×1) process state vector at time t_k, $\boldsymbol{\varphi}_k$ is (n×n) state transition matrix, $\mathbf{w}_k$ is (n×1) vector, assumed to be a white sequence with a known covariance structure, $\mathbf{Z}_k$ is (m×1) measurement (observation) vector at time t_k, $\mathbf{H}_k$ is (m×n) matrix giving the ideal (noiseless) connection between the measurement and the state vector at time t_k, and $\mathbf{v}_k$ is (m×1) measurement error, assumed to be a white sequence with known covariance structure and having zero cross-correlation with the $\mathbf{w}_k$ sequence.

If this two-dimensional model works properly for surface based UAV applications, such as airport taxing, factory floor or parking lot, the three-dimensional model will be built in the next step to resolve conflicts in air space.

Conclusion

The presented research is the first step towards making the concept of Extended GLA a useful tool for modeling and simulation, as opposed to pure implementation of hybrid intelligent control systems. Thus far, extending GLA by adding three more orthogonal components at the Process Layer level turned out to be useful for specific applications considered, as confirmed by simulation experiments.

Further work is needed on extending GLA toolset to support the design process with appropriate user interfaces and network and database connectivity. This will significantly enhance the development process currently supported mostly by off-the-shelf tools.

Acknowledgements

Some of the ideas described in this paper have been developed within master-level projects at the Department of Computer Science of Lund University (GLA), and at the Computer Engineering Program of the University of Central Florida (real-time software architecture).

References

Al-Daraiseh, A., Al Mazid, A., Croeze, T., Zalewski, J., Dolezal, M., Green, G., Bahr, H. (2000). High-Level Tool Support for Integration Architecture in a Distributed Embedded Simulation Project, *Proc. CSMA2000, 2nd Conference on Simulation Methods and Applications*, Orlando, Florida, October 27-29, pp. 33-36

Grossman, R.L., Nerode, A., Ravn, A.P., & Rischel, H., editors. (1993). *Hybrid Systems*, Berlin: Springer-Verlag.

Guo, D., van Katwijk J., Zalewski, J. (2003). A new benchmark for distributed real-time systems: Some experimental results, *Proc. WRTP'03, 27th IFAC/IFIP Workshop on Real-Time Programming*, Łagów, Poland, May 14-17.

Johnson, R., Sasiadek, J., Zalewski, J. (2003). Kalman Filter Enhancement for UAV Navigation, *Proc. SCS 2003 Collaborative Technologies Symposium*, Orlando, Fla., January 19-23, pp. 262-272.

Lin, M. (1999a). *Analysis and Synthesis of Reactive Systems: A Generic Layered Architecture Perspective*, PhD thesis, Department of Computer Science, Linköping University.

Lin, M. (1999b). Timing analysis of {PL} programs. *Proc. WRTP'99, 24th IFAC/IFIP Workshop on Real-Time Programming,* Dagstuhl, Germany, May 30 – June 3.

Lin, M., & Malec, J. (1998). Timing analysis of RL programs. *Control Engineering Practice*, *6*, 403-408.

Malec, J. (1992). Complex behavior specification for autonomous systems. *Proc. IEEE International Symposium on Intelligent Control'92*, pp. 178-183, Glasgow, Scotland,.

Malec, J., Morin, M., & Nadjm-Tehrani, S. (1995). A layered software architecture for design and analysis of embedded systems. *Proc. 1995 International Symposium and Workshop on Systems Engineering of Computer Based Systems*, pp. 169-176, Tucson, Ariz.

Mathure M., Jonnalagadda V., & Zalewski J. (2003). Heterogeneous architecture and testbed for simulation of large-scale real-time systems. *Proc. 7th IEEE Int'l Symp. on Distributed Simulation and Real-Time Applications*, pp. 37-42, Delft, The Netherlands.

Morin, M. (1991). *PLCL - Process Layer Configuration Language.* Technical Report LAIC-IDA-91-TR10, Linköping University.

Morin, M. (1993). *Predictable cyclic computations in autonomous systems: A computational model and implementation*. Licentiate thesis 352, Dept. of Computer and Information Sciences, Linköping University.

Morin, M. (1994). *RL: An embedded rule-based system.* Technical Report LAIC-IDA-94-TR2, Linköping University.

Morin, M., Nadjm-Tehrani, S., {\"O}sterling P., & Sandewall E. (1992). Real-time hierarchical control. *IEEE Software*, *9*(5), 51-57.

Sanz, R. & Zalewski J. (2003). Pattern-Based Control Systems Engineering, *IEEE Control Systems*, *23*(3), 43-60.

Zalewski J. (2001). Real-time software architectures and design patterns: Fundamental concepts and their consequences, *Annual Reviews in Control*, *25*(1), 133-146.