Model-Based Approach for Intelligent Control

Janos Sztipanovits, Csaba Biegl, Gabor Karsai
Vanderbilt University, Nashville, TN 37235

R.Byron Purves
Boeing Aerospace Company, Huntsville, AL 35807

ABSTRACT

The paper discusses a comprehensive, model-based approach for the design and implementation of intelligent controllers. The system has been implemented in the framework of the *Multigraph Architecture.* The Multigraph Architecture is a layered system, which includes a parallel, graph computation model, the corresponding execution environment, and software tools supporting the interactive, graphical building of knowledge-bases.

1. INTRODUCTION

The design of large-scale automation systems that must operate in unstable, changing situations is one of the foremost challenges of the information sciences. Conventional design methodologies are based on the availability of a priori information about the environment and the system to be observed and controlled. The information is expressed in the form of models representing relevant aspects of the environment. The basic modeling principles of the system sciences such as *separation*, *selection*, and *model economy* [1] are the key approaches for managing complexity. The essence of these principles is *simplification* until a model of manageable size is obtained. By imposing constraints on the possible behavior of the environment, the analysis and/or synthesis of the corresponding automation system becomes feasible.

The critical issue in this methodology is what to do if the constraints suddenly do not hold? Such situations may occur as the consequences of perturbations in the environment, or as the results of catastrophic system component failures. Possible approaches to this problem are the followings:

- stabilize the environment so that this cannot happen,
- design robust systems that are insensitive to any kind of changes,
- develop systems that can detect changes in the environment, analyse their impact on the control system, and take corrective actions in the system operation.

From the choices listed above, the last one is the only feasible approach for a large class of problems. However, the implementation of control systems that exhibit the required level of adaptivity is not straightforward. The main problem is that in abruptly changing, unstable environments parameter adjustments may not provide enough flexibility for adapting the system operation; the system structure should be modified [2].

Systems that are able to modify their structure in order to• adapt to changes in the environment have the utmost importance. While previous work has identified many of the problems and offered solutions for particular issues, no comprehensive approaches have thus far been developed. Extensive research, particularly in the field of *intelligent control* has been addressing these problems. Important directions are *reconfigurable control*, *expert controllers*, and *neural controllers.*

- *Reconfigurable control systems* address the problem of ever-changing environment and/or process states by operating several controllers in parallel and choosing among them on-line when a change in the system state is detected. Examples for this approach are the MIT reconfigurable controller developed for the 5MW Research Reactor [3], or the controller developed for the High Flux Isotope Reactor at ONL [4]. The main strength of these systems is their capability for on-line reconfiguration, but they have strong limitations in their adaptivity because the number of structurally different controllers that can operate in parallel is restricted.

- *Expert controllers*, in a broader sense, cover the efforts for applying expert systems in control. From the point of view of adaptive behavior, the closed-loop, rule-based controllers represent an interesting research direction, The basic structure is quite similar to the fuzzy controller approach [5]. In most of the experimental systems a new "higher-level" is introduced by using rule-based expert system techniques. The role of the expert system component in these controllers is to allow the use of heuristics in the control loop for tuning the controller [6] or for directly executing control actions [7]. Although the potential for using expert systems as a higher-level organizer and decision maker in adaptive controllers has been mentioned, e.g., in [81 and [9], there is no reference to the actual implementation of a controller where the expert system would make on-line changes in the structure of the low-level controller.

- The latest development in this area is the application of *neural networks* for control (see e.g. a recent special section of the IEEE Control Systems Magazine [10].) The experimental architectures include proposals for the neural network to be applied as a feedforward controller or to be included in an adaptive control scheme for identifying the state of the plant. The common element in these proposals is that the systems try to take advantage of the learning capabilities of the neural networks.

Model-based methodologies have great potential in implementing structurally adaptive controllers. The main idea is quite straightforward and includes the following steps.

- A dynamic model of the environment (the system to be observed or controlled) is included in the signal processing or control system.

- The model is continuously updated based on observations.

- The control system is modified (structure and parameters) if state changes in the model require it,

This paper will focus on the computational problems of creating structurally adaptive controllers by using model-based techniques. The purpose of the discussion is to show the key components of a programming and execution environment that can be used for implementing this new system category.

2, STATEMENT OF THE PROBLEM

The main computational requirements in the implementation of structurally adaptive controllers are the followings:

- The dynamic model of the environment and its interactions with the structure of the control system must be represented.

- The representation must be used as part of the adaptation process, i.e. changes in the environment model must be mapped into changes in the structure of selected system components.

- The structural changes must be executed without suspending the system operation.

By using artificial intelligence terminology, the first requirement creates a *knowledge representation* problem. Naturally, the model-based approach demands the explicit representation of models. The key issue is what kind of representation techniques can be used for this purpose? The second requirement addresses the problem of *knowledge utilization.* The knowledge which represents the interactions between the environment and the structure of the control system has to be actively used for modifying the system operation. The problem is how to "convert" this knowledge dynamically into implementation specific terms? The third requirement is closely related to the *computational model* used in the execution environment of the control system. The question is what kind of computational model can support the dynamic reconfiguration of a processing system in execution time?

The main difficulty in the technology of structurally adaptive, intelligent systems is that realistic implementation can not be built without finding satisfactory solution for each of these problems. A detailed analysis and abstract formulation of the computational requirements has been given in [11]. In the followings we will focus on the

description of the components of the Multigraph Architecture which has been designed to serve as a generic programming and execution environment for this system category.

3. MULTIGRAPH ARCHITECTURE

The Multigraph Architecture (MA) has been developed for building a broad category of intelligent systems operating in real-time environment. The MA has been used as a framework for intelligent instrumentation [12], automatic test configuration [13], and process control [141 systems. The description of the basic layers of the MA, namely: (1) *hardware layer,* (2) *system layer,* (3) *module layer,* and *(4) knowledge layer,* are given in 115]. In Figure 1, the three main levels of the MA are shown from the user's point of view.

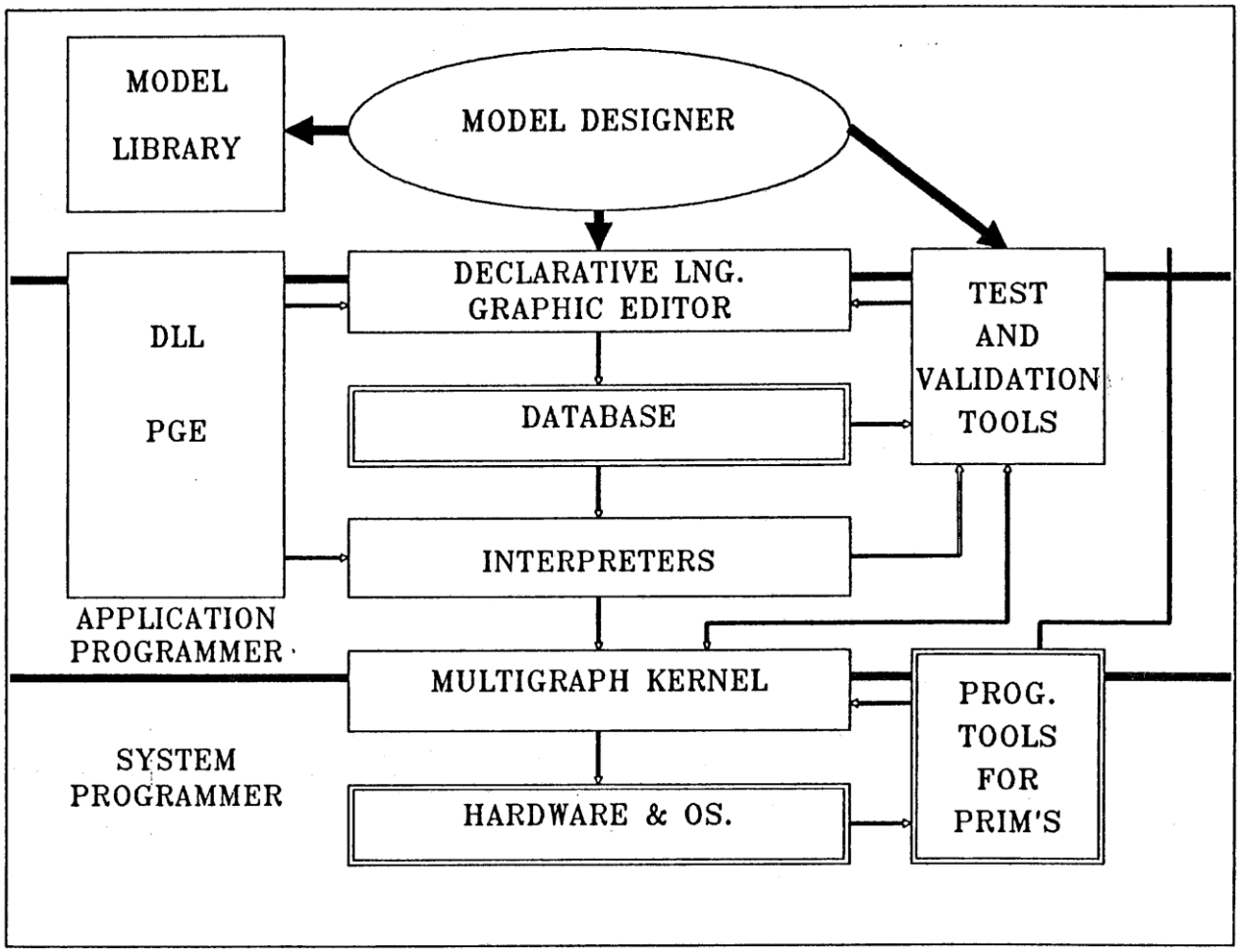

Figure 1. Structure of the Multigraph Architecture

- *Model Designer.* The design and implementation of model-based, intelligent controllers requires extensive modeling. Because the adaptation process may require structural modifications in the control system, the models must be hybrid. Hybrid models explicitly represent not only quantitative, but qualitative, structural attributes of the environment and the control system. Model designers must be supported by appropriate tools to build and validate these models.

- *Application Programmer.* The models that are used in the design and implementation of intelligent controllers are domain specific by their very nature. The form of the models (concepts, relationships) are different in chemical processes, mechanical processes, information processing systems etc., because the models must reflect the selected properties of these systems. However, some of the basic modeling principles, such as composition

techniques, organization in levels of abstraction, multiple-aspect representation, etc. are quite universal. This generality makes it possible that the creation of domain specific modeling tools can be supported by general methodologies. The application programmer level in MA includes those components that are used for building various, domain specific modeling environments.

- *System Programmer.* The lowest level of MA provides interfaces to the components of the Multigraph Execution Environment (MEE). The central element of MEE is the Multigraph Kernel (MK), which is the run-time support of the Multigraph Computational Model (MCM). MCM is a macro-dataflow model which satisfies the required dynamic behavior mentioned before.

4. DECLARATIVE/GRAPHIC PROGRAMMING ENVIRONMENT FOR MODEL BUILDING

The models that are created during the modeling process are complex structures representing different aspects of the environment, the control system and their interactions. It is important to note that in these models the structural complexity is the dominant factor, the algorithmic complexity is typically negligible. This fact had deep influence on the properties of the Multigraph Programming Environment (MPE). The two basic techniques used for supporting this activity are (1) multiple-aspect model building and (2) declarative/graphic programming.

- *Multiple-aspect model building.* Characterization of objects from different aspects is a well known method in modeling. There are artificial intelligence (Al) tools (e.g. ART, [16]) that directly support the creation of "multiple views". According to our experiences, the real difficulty is not the representation of different aspects but the expression of the interactions among them. The critical question is how to facilitate the well, structured representation of these interactions? MPE allows the declaration of *structurally independent* (SI) and *structurally dependent* (SD) modeling aspects. Examples for SI aspects are the *physical model* of a plant describing its component hierarchy, and its *process model* representing the dynamic, physical interactions (energy, material transfer processes) in the system. The interrelationships among the SI aspects can be expressed by defining conceptual links among their elements. For example, links can be used to define what are the physical components of a plant that are directly involved in a heat-exchange process. SD aspects are embedded in the model of a so called *dominant aspect.* In the process control domain, we can consider the *process structure* as a dominant aspect, and *the fault model, fault detection model, control* model and *operator interaction model* as SD aspects. The process model defines on each level in the process hierarchy the input/output variables of the processes, the list of the internal process variables, the constraints among these variables, and the internal structure of the processes i.e. the sub-processes and their interactions. Obviously, all of the SD modeling aspects are closely related to this model. For example, the fault propagation models can be defined in terms of the fault modes of processes and their propagation characteristics. The fault detection models use quantitative/qualitative relationships among the process variables to detect various fault modes. The control models link the measured and controlled process variables and the operator interaction can also be structured according to the levels in the process hierarchy.

- *Declarative/graphic model building tools.* Modeling requires tools for representing the models. The representation technique has to satisfy two contradictory requirements. First, the representation system must provide "interface" for the model designer, i.e. the represented model has to be easily comprehensible by humans. Second, the represented model has to be machine readable, because the models constitute the "knowledge- base" which determines the system operation. Based on these requirements and on the fact that the models express dominantly structural information, MPE supports two equivalent representation form: *declarative languages* and the corresponding *graphic representation.* The model building process, which is performed by the model designers is fully graphical and directly supports SD and SI modeling. For example, after starting the Fault Model Editor, an initial version of the fault propagation graph is automatically extracted from the process model based on the already defined process structure (sub-processes and their interactions). This initial model will be further refined by the model designer. The result of the graphical model building process is not only the graphic image of the model, but the declarative language representation as well. The declarative language representation is automatically generated by the graphic editors.

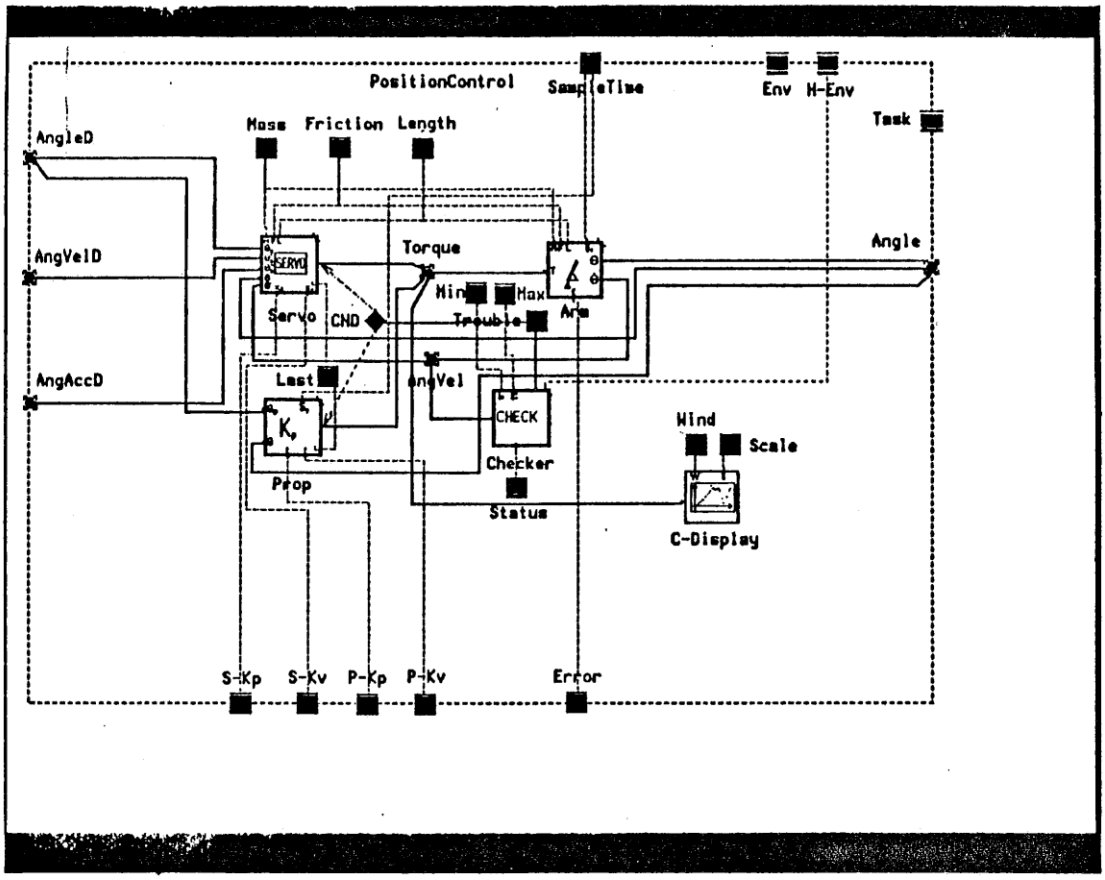

Figure 2. Graphic Model of a Reconfigurable Controller

- Figure 2 shows the graphic model a reconfigurable controller for a simple robot arm. The arm is controlled by (a) a proportional controller, or (b) a PID controller. The reconfiguration occurs when the "Checker" finds the performance of one of the controller unacceptable. The figure shows only the top level structure of the controllers and the simulation model of the arm, each of the boxes have an internal structure on the lower levels of the hierarchy. The graphic model has been built by using the iconic editor of MPE. Figure 3 shows a part of the equivalent declarative language representation of the model. The declarative language is a variation of the "frame languages", which can be easily defined for the different modeling domains.

- *Test and Validation Tools.* Declarative languages offer excellent opportunity for automatic test and validation [17]. The basic approach used in the test and validation toolset of MPE includes the following steps: (1) the declarative language forms are mapped into a unified graph structure, (2) test and validation criteria are defined for the different modeling aspects, (3) the criteria are expressed as graph properties, and *(4)* graph algorithms are used to check the properties. The methodology supports the automatic test of the consistency of the individual modeling aspects and the consistency among the SD aspects. A serious limitation of the test approach is that only static properties of the models can be tested this way. In a new research direction we address the problem of testing the dynamic, run-time behavior of the system.

An important goal of MPE is to facilitate the definition of declarative languages and the corresponding graphic editors for new application domains. Generic tools belonging to the level of the Application Programmer support this task which includes the following steps: (1) definition of the syntax of the declarative languages, and (2) configuration of the corresponding graphic editor. The two programming tools developed for this purpose are the Declarative Language Language (DLL), and the Programmable Graphic Editor (PGE), respectively. Detailed description of these tools are given in [17].

5. SYSTEM INTEGRATION

Multiple-aspect models of the external environment (platforms, signal sources, etc.), the various components of the control system (monitoring systems, controllers, etc.), and their interrelationships embody the information that is necessary to generate a specific *instance* of the structurally adaptive control system. The problem of system integration is to generate this instance from the models, or in other words, to map the models into an appropriate executable program. Because of the implementation method of this mapping, we will call this process *model interpretation.*

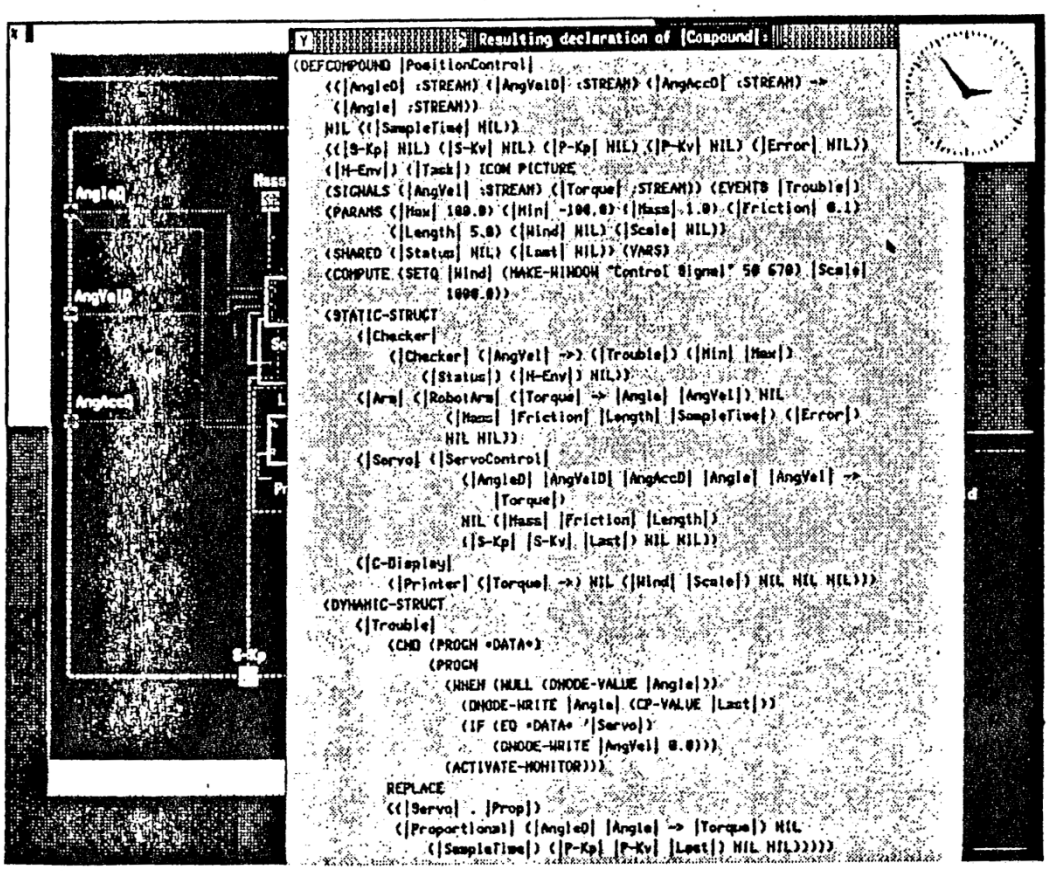

Figure 3. Declarative Language Representation of the Controller

The complexity of the model interpretation process largely depends on the nature of the models. If it includes only the symbolic, static model of a specific system, e.g. the model of a controller, the model interpretation process is reduced to the complexity of simple application generator systems (see e.g. [19]). In the general case, the structurally adaptive controllers require the following capabilities from the model interpretation process.

- *Multiple-aspect interpretation.* The result of the model interpretation process must generate more than one 'subsystems of the controller. For example, an "intelligent process control system" includes the physical interface, monitoring system, controller, fault diagnostics and operator interface subsystems [141. Multiple-aspect model interpretation means that the mapping process must interpret the models *from the aspects of the various subsystems* to be generated.

- *Decision making.* The complexity of the mapping process is largely the consequence of the fact that the models are not structured according to the subsystems of the system to be generated. (Except the simple application generator problems, where modeling is usually constrained to specific computation systems to be generated,) Indeed, in model building time the natural way of thinking is to focus on selected aspects of the environment,

the control system and their interactions *without* any explicit considerations to the actual way of implementation. The model interpretation process has to be "smart enough" (1) to coiled the relevant information from the models for the various subsystems, and (2) during this process to make decisions on the actual structure of the computation system by analysing the interaction of the different modeling aspects.

- *Dynamic behavior*. The essence of any structurally adaptive system is the capability for dynamic reconfiguration of subsystems after a change in the environment has been detected. It means that the model interpretation process has to be restartable from that point which has been effected by the detected change.

These capabilities required the elaboration of a special computation model in the Multigraph Execution Environment (MEE) described in the next section. Important elements of the technology are the model interpreters. Because different interpreters are needed for different application domains, the construction of interpreters is supported. Application programmer level libraries facilitate the implementation of model interpreters, and provide a general framework for their design.

6. EXECUTION ENVIRONMENT

MEE implements the run-time execution environment in the system architecture. It provides a system integration tool by supporting the dynamic configuration of application programs from a library of precompiled elementary processing modules. This configuration process can be performed by the higher-level knowledge-based system components using an appropriate builder interface of the MEE. Frequently the usage of the MEE also enables the utilization of the inherent structural parallelism in the application programs, since it is quite typical that many of the processing modules of an application configured using the above method can be executed concurrently, provided that the underlying hardware architecture supports this.

MEE uses a macro-dataflow model as its basic computational model [20]. The reasons for this choice were (1) the well-known nature of the dataflow computations due to the significant amount of research conducted on exploring the theoretical properties and implementational issues of these, and (2) the fact that many engineering system models (for example the signal flow graphs used in signal processing and process control systems) can easily be mapped into dataflow graphs. Some extensions were added to the "typical" dataflow computational concepts, because the MEE serves as a unified run-time support for the different parts of the intelligent process control systems, and these parts might use different models of computation (for example signal-flow graphs, discrete event simulators, rule interpreters, constraint propagation networks, etc..).

The applications in the MEE are mapped into a *control graph*. A control graph in the MEE is defined by its *actornodes*, *datanodes* and *connection specifications*. The actornodes are the active components of the graphs. They execute an application module (the script) which can be written either in Lisp or in other non-symbolic languages (C, Fortran, Pascal). The scripts are position independent, they communicate with the other graph components using the communication primitives of the MEE and the ports attached to the actor node. If the code of the script is reentrant, it can be attached to several actornodes. The MEE provides a way to pass a local parameter structure to the scripts, which is called the context of the actornode. Beside the typical dataflow control principle (a node can be fired whenever all of its inputs are present - *ifall* mode) MEE also supports another mode of actornode execution, where a single input data is enough to fire a node *(ifany* triggering mode).

The datanodes are the passive components of the control graphs. Their function is to store the data generated by the actornodes. They can store either a stream of data, or only the last data sent to them.

MEE supports several operation modes of a control graph. A graph can be operated either in *data-driven* or *command-driven* mode, or in a combination of the two modes. In the data-driven mode, the data sent to a datanode propagates a control token to the following actornodes, which will fire after collecting the necessary tokens. The command-driven mode means that an attempted read operation on an empty datanode will send a request token to all possible sources (in. the connected actornodes) of the information.

MEE provides an environment and task structure which is used to assign the various system resources of the system hardware and software (processors, tasks, special hardware units, etc.) to the execution of the actornodes in the computational graphs.

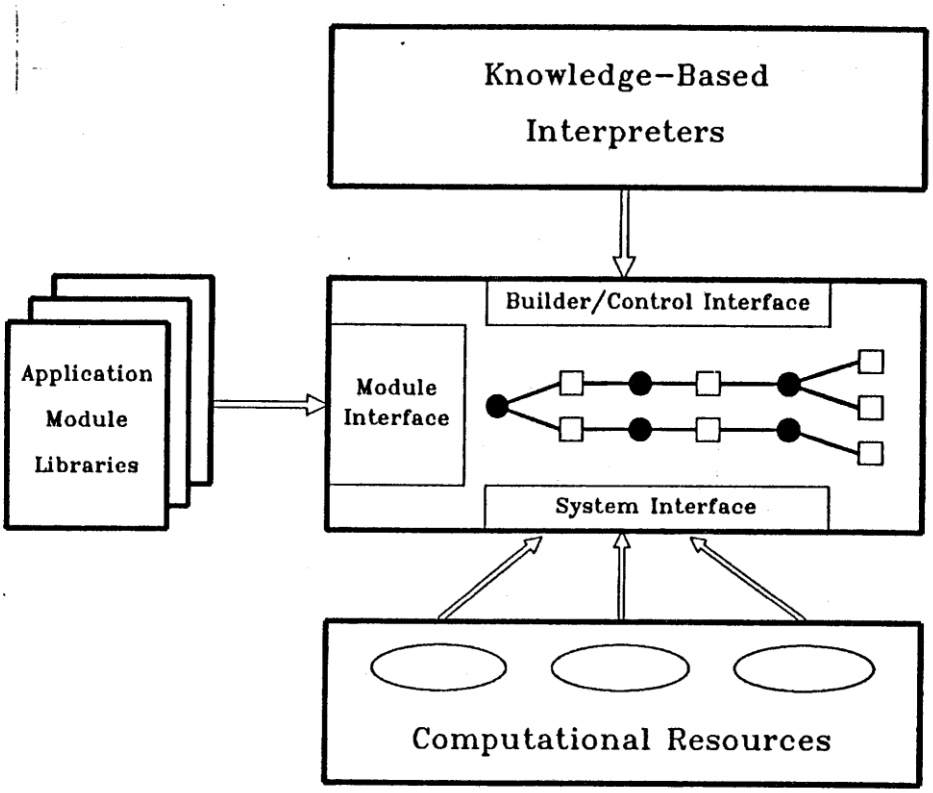

Figure 4. Structure of the MEE

The structure of a typical implementation of the MEE can be seen in Figure 4. MEE can be depicted as a set of protected data structures which can be accessed through the following three interfaces:

- *Module Interface:* which provides the data and request propagation calls for the application modules attached to the actornodes.

- System Interface: which is responsible for scheduling the elementary computations using the system resources provided by the host operating system.

- Builder and Control Interface: which provides the control graph building and execution control facilities for the higher-level knowledge-based system components. The services of this interface can operate on an already active computational graph, which enables the *dynamic reconfiguration* of the application programs.

The control graph of the reconfigurable robot arm controller can be seen in the upper right window of Figure 5. The larger rectangles represent the actornodes, the smaller ones the datanodes. The number of actornodes are larger than the number of blocks in Figure 2, because some of the blocks are compound structures. The left windows in Figure 5 monitor the operation of the controller.

MEE offers a set of debugging tools which are especially helpful in concurrent systems. These include a stepper/tracer facility and a graphic monitor, which generates and displays the graphic layout of selected parts of the control graph, and dynamically displays the status of the nodes in the graphic window.

The computational model and the details of its implementation were selected such that the Kernel can provide the same execution environment on a variety of computer architectures, by hiding the details of the (possibly parallel) execution from the application modules, which can be simple sequential procedures in every case.

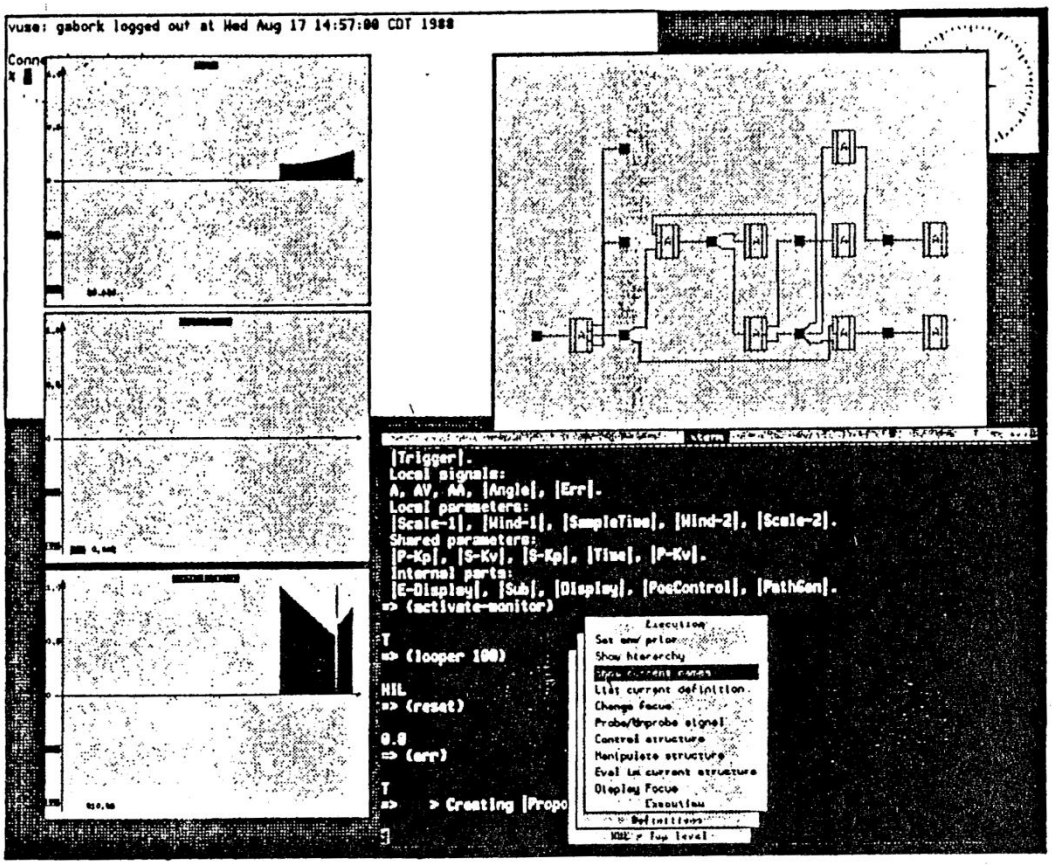

Figure 5. Control Graph of the Reconfigurable Controller

7 CONCLUSION

The paper described the components of a comprehensive, model-based approach for the design and implementation of intelligent controllers. The system has been implemented in the framework of the *Multigraph Architecture* which includes a parallel, graph computation model, the corresponding execution environment, and software tools supporting the interactive, graphical building of knowledge-bases. The paper has shown that the problems of knowledge representation, interpretation and program execution are very closely related in intelligent controllers where the dynamic reconfiguration of the system is a primary requirement.

REFERENCES

1. Karplus, WJ.: "The Spectrum of Mathematical Modelling and Systems Simulation," *Mathematics and Computers in Simulation,* Vol. 20, No.1., pp. 3-10, 1977.

2. Sztipanovits, J.: "Toward Structural Adaptivity," *Proc of the 1988 IEEE International Symposium on Circuits and Systems,* Espoo, Finland, pp. 2359-2362, 1988.

3. Ornedo,. R.S., Bernard, J.A., Lanning, D.D., and Hopps, J.H.:"Design and Experimental Evaluation of an Automatically Reconfigurable Controller for Process Plants", *Proc. of American Control Conference,* pp.1662-1665, Minneapolis, MN, 1987.

4. Clapp, N.E., Clark, F.H., Mullens, LA., Otaduy, D.J., Wehe, D.K.: "Application of Expert Systems to Heat Exchanger Control at the 100 MW High Flux Isotope Reactor", *Proc. of 1985 Controls West Conference,* p.208, Tower Conference Management Co., Wheaton. IL, 1985.

5. Mamdani, E.H., Assilian, S.: "An Experiment in Linguistic Synthesis with a Fuzzy Logic Controller," *Int. Journal of Man Machine Studies,* Vol 7., p.1, 1975.

6. Moore, R.L. et. al.:"Expert System Applications in the Industry," *Proc. of the ISA International Confere.ice,* pp. 22-25, Houston, TX, 1984.

7. Canon A.:"Considerations in the Application of Self-Tuning PID Controllers Using EXACT-tuning Algorithm," *Measurement+ Control,* Vol. 19, pp. 260-266, 1986.

8. Astrom, K.3.: "Auto-Tuning Adaptation and Expert Control," *Proc. of the 1985 American Control Conference,* pp. 1514-1519, *1985.*

9. Rodin, E.Y.:"Semantic Control Theory," *Applied Math. Lett.,* Vol. 1, No. 1, pp.73-78, 1988.

10. IEEE Control Systems Magazine, April 1988.

11. Sztipanovits, 3., Karsai, G., Biegi, C.:"Modeling, Model Interpretation and Intelligent Control," *Proc. of the 3rd. IEEE International Symposium on Intelligent Control,* Arlington, Virginia, 1988. (in press)

12. Sztipanovits, J., Biegi, C., Karsai, G., Bourne, J., Mushlin, R., Harrison, C.: "Knowledge-Based Experiment Builder for Magnetic Resonance Imaging (MRI) Systems," *Proc. of the 3rd IEEE Conference on Artificial Intelligence Applications,* Orlando, FL, pp. 126-133, 1987.

13. Sztipanovits, 3., Purves, B.R.: "Coupling Numeric and Symbolic Computations in Real-Time, Distributed Computing Environment," in Kowalik, C. (ed.) *Coupling Symbolic and Numeric Computations,* North-Holland Publishing House, pp. 117-128, 1988

14. Karsai, G., Biegi, C., Padalkar, S., Sztipanovits, J., Kawamura, K., Miyasaka, N., Inui, M.: "Knowledge-Based

Approach to Real-Time Supervisory Control," *Proc. of the l988Amencan Control Conference,* Atlanta, Georgia, pp.620-626, 1988.

15. Sztipanovits, J.: "Execution Environment for Intelligent Real-time Control Systems," *Proc. of the NASA/JPL Symposium on Telerobotics,* Pasadena, CA, pp. 131-139, 1987.

16. Clayton, B.C.:"ART Programming Tutorial", *Inference Corp.,* 1985.

17. Sztipanovits, J., Padalkar, S., Krishnamurthy, C., Purves, R.B.:"Testing and Validation in Artificial Intelligence Programming", *Proc of the 3rd SPIE's Conference on Space Station Automation,* Cambridge, MA. 1987. (in press)

18. Karsai, G.: "Declarative Programming Techniques for Engineering Problems", Ph.D. Thesis, Vanderbilt University, 1988.

19. Rich, C., and Waters, R.C.:"Automatic Programming: Myths and Prospects", *Computer,* pp.40-Si, August 1988.

20. Biegl, C.: "Design and Implementation of an Execution Environment for Knowledge-Based Systems", Ph.D. Thesis, Vanderbilt University, 1988.