Model-Integrated Parallel Application Synthesis

Akos Ledeczi
Department of Electrical and Computer Engineering, Vanderbilt University
akos@vuse.vanderbilt.edu

Abstract

Parallel computer architectures based on such processors as the Texas Instruments TMS320C40, or the Analog Devices ADSP21060, are characterized by high performance and I/O bandwidth, flexible topology and low cost making them ideal for embedded parallel signal processing and instrumentation applications. However, the software of such systems is difficult to manage by conventional software engineering methods because of the complexity of the large-scale parallel application and the flexibility of the hardware topology. This paper discusses the adoption of a model-integrated programming environment, the Multigraph Architecture (MGA), to this domain. Using the MGA, the parallel application is automatically synthesized from high-level system models and assigned to the available network of processors.

1. Introduction

Parallel computer architectures based on such processors as the Texas Instruments TMS320C40, or the Analog Devices ADSP21060, are characterized by high performance at low cost. These systems are highly scalable, flexible and modular. The topology of the network can be arbitrary, limited only by the maximum degree of the nodes. The interconnection architecture can be designed specifically for the given application. The size of the network can be easily adjusted as the system requirements change. Nodes can be added one-by-one at any location in the network with an available communication link. Furthermore, the multiple, high-speed communication links of the processors can be used to interface to external devices achieving high I/O bandwidth.

These favorable characteristics make these processors ideal for embedded parallel signal processing and instrumentation applications. Systems can contain from a couple of processors up to hundreds of nodes connected by an interconnection network with flexible topology. They are able to process data on multiple channels at high data rates in real-time. However, the software of such systems is difficult to manage by conventional software engineering methods because of the complexity of the large-scale parallel application and the flexibility of the hardware topology.

Model-integrated programming is a promising new software technology that is able to address the issues associated with these systems. The Multigraph Architecture [1-2] is a model-integrated programming environment that has been applied successfully in diverse fields, including process monitoring and control, fault detection, isolation and recovery, and discrete manufacturing. It suits parallel processing well. The graphical, multiple-aspect, hierarchical system models manage the software and hardware complexity of the application, while the automatic system synthesizer and the run-time environment provide process synchronization and communication transparently. Embedded parallel instrumentation and signal processing is a relatively new area lacking a mature software engineering technology. Model-integrated automatic system synthesis is a promising technology that has great potential in this domain. Applying the Multigraph Architecture to embedded signal processing and instrumentation on distributed memory multiprocessors with flexible interconnection topology is in the focus of this paper.

2. The Multigraph Architecture

The Multigraph Architecture (MGA) provides a unified software architecture and tools for: (1) building, testing, and storing multi-aspect, graphical domain models, and (2) transforming the models into executable programs and/or extracting information for system engineering tools [1-2]. The MGA has the following functional components (Figure 1):

- Graphical Model Builder (GMB). The modeling paradigm supported by the GMB includes concepts, relationships, model composition principles, constraints, and representation techniques that are accepted and used in the application domain. The GMB tool provides a customizable model building environment for domain experts. It enforces domain-specific constraints during model building, uses domain-specific graphical formalism, and supports checking the models against consistency and completeness criteria.

- Model Database. The model database stores the complex, multiple-aspect models. Typically, off-the-shelf or public domain object oriented databases are used for this purpose.
- Model Interpreters. Model interpreters synthesize executable programs from domain models and generate data structures for system engineering tools that perform various analyses of the systems to be built. Since the model interpreters capture the relationship between the problem space and the solution space, they are specific to the domain.
- Multigraph Kernel. The executable programs are composed in terms of the Multigraph Computational Model (MCM). The MCM is a macro-dataflow model providing a unified system integration layer above heterogeneous computing environments, including open system platforms, parallel/distributed computers, and signal processors. The run-time support of the MCM is the Multigraph Kernel (MGK). The MGK provides scheduling, synchronization, and communication. The elementary computations are carefully defined reusable code components that are part of application specific run-time libraries. The model interpreters synthesize the applications by building the dataflow graph and setting the parameters of the elementary computation blocks.

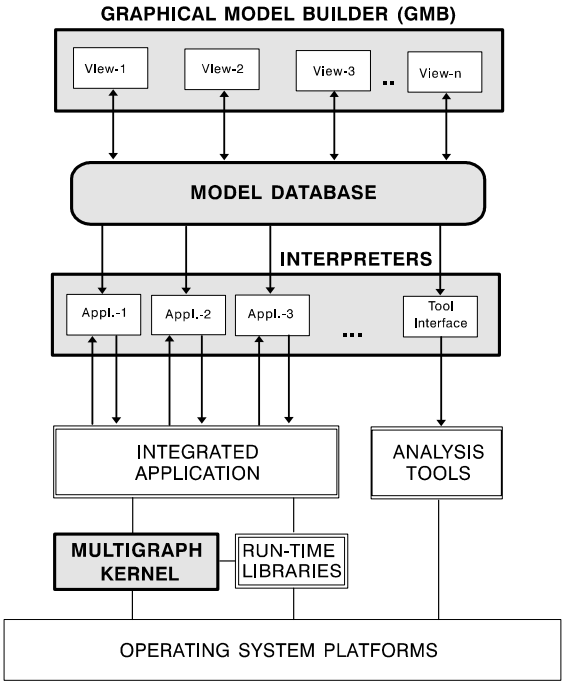

Figure 1. The Multigraph Architecture

3. Problem Statement

The objective of the research is to develop a framework for automatic synthesis of large-scale, parallel instrumentation and signal processing applications characterized by high I/O bandwidth, computationally intensive processing requirements, and frequently changing software specifications and hardware configurations. The target hardware platform is distributed memory multiprocessors with flexible interconnection topology. To achieve this goal, the following issues are addressed:

- Representation. The application specifications must be represented in a computer readable format to facilitate automatic application synthesis. Furthermore, the representation format must be easily comprehensible by humans. In order to manage the complexity introduced by low level parallel processing and systems engineering issues, high level system descriptions are needed. The specifications must include the application requirements and the available software and hardware resources. The representation technique must provide means to manage the complexity of the specifications themselves.
- Automatic application synthesis. The parallel instrumentation application must be automatically synthesized from the high level system specifications. The software system needs to be partitioned and assigned to the hardware platform. Executables, message routing information, and network loader configuration are to be automatically generated. The specific requirements of the process assignment and the message routing strategy are as follows:
- Process assignment. Process assignment must be carried out automatically in order to optimize the performance of the synthesized system. A cost function is needed that accurately describes the quality of the assignment. Locating the optimal solution cannot be guaranteed because the problem is NP-complete. The search space must be restricted to keep system synthesis time polynomially bound.
- Message routing. Deadlock-free wormhole routing in networks with arbitrary topologies is an open problem. Deadlock-freedom must be guaranteed. Minimizing the communication overhead is critical to the performance of the system.

The following restrictions are placed on the problem domain to keep the research well focused and the problems in the preceding list manageable:

- Signal flow dominance. The class of targeted applications are limited to signal flow dominant systems. The structure of such systems can be described by a signal flow graph.
- Static structure. The signal flow graph of the system is static. Dynamic reconfiguration is not permitted.

- Continuous execution. The execution of the signal flow graph is continuous. Processing of consecutive input sets overlap in a pipeline fashion.
- Throughput. The objective of the system synthesis process is to maximize system throughput. Real-time constraints are not considered.

The solution domain is restricted by the following factors:

- Task parallelism. The data parallel computational model is not considered.
- Hardware platform. The target hardware platform is distributed memory multiprocessors with flexible interconnection topology.

The problem of automatically synthesizing large-scale, parallel instrumentation and signal processing applications for distributed memory multiprocessors with flexible interconnection topology is solved in the framework of the Multigraph Architecture (MGA). In order to manage the high complexity of the system models, the declarative modeling capabilities of the MGA are augmented by an additional model organization principle: generative model building. The parallel instrumentation domain mandates three modeling aspects: signal flow, hardware, and assignment constraints aspects.

It is the task of the model interpreters to partition the signal flow graph and assign the partitions to the nodes of the processor network while satisfying the assignment constraints. A deadlock-free wormhole routing strategy for networks with arbitrary topologies is developed and integrated into the model interpreters. The output of the MIPAS includes executables, a dataflow graph partition, and a message routing map for each processor in the system.

4. Modeling Paradigm

The Multigraph Architecture (MGA) supports declarative modeling of complex systems. It provides several model organizational principles to manage the complexity of the system models. Multiple aspects, model types and instances support modular modeling. Model references aid in the description of interactions between modeling aspects. Hierarchy and multiple views provide visibility control. While these are powerful techniques that help the management of the complexity of the system models, experience shows that there is a clear need for an additional model organization principle primarily for modeling repetitive structures. The following section describes how generative modeling can satisfy this need, and how it can be incorporated into the declarative modeling environment of the Multigraph Architecture.

4.1. Generative Modeling

When several parts of a model have the same components and structure, simple replicators could reduce the complexity of the models. Instead of repetitively building the same model for every occurrence, one copy and the desired number of replications could be specified. However, the interface of such replicators poses problems. Since there is only one actual copy of the model, only one connection can be made to each of its ports. Such a connection could be interpreted as one connection to each replicated instance or as a single connection to the first instance. Some complicated constructs could be defined for different cases, but the solution would not be intuitive and easy to use. A situation similar to that of the replicators exists with conditional model components whose existence depend on some condition.

Conditional models are very useful for modeling complex systems. For example, changing requirements and varying resources may force the user to change the system models frequently. The required "size" of the system changes most often. For instance, the number of channels required in a multi-channel system can vary from day to day. Similarly, the number of available hardware resources, such as processing nodes, disk drives, printers, etc., can also change frequently. Editing the system models often is cumbersome and error-prone. A simple solution is to model the biggest expected configuration and conditionalize parts of it. Conditionals are similar to replicators because they specify the number of occurrences of a model component, which can be zero or one. These two modeling constructs, replicators and conditionals, can be combined and implemented with generative modeling.

With generative modeling, the user can specify model structure, i.e. components and connections, by writing a program in some language. Generative modeling is similar to the generate statements of VHDL [3]. To interface this style of model building to the declarative (graphical) modeling paradigm of the MGA, the textual attribute feature of the modeling environment is utilized.

Each component of a Multigraph model can have multiple textual attributes to capture information that cannot be represented graphically. A varying number of textual attributes are dedicated to generative modeling depending on the type of the model component. Model components with inner structure have a structure attribute that is used to create new, or destroy existing connections between parts of the given model. Every model component has a repetition attribute, which expresses the number of repetitions of the current model. A reference attribute is assigned to models that contain references to components in other aspects. Since model references can be made only to graphically specified model components, this attribute is used to refer to components specified by generative modeling.

These textual attributes are called generative attributes. The language selected for generative modeling is C++ because it is widely used and compilers are readily available.

The primary modeling methodology in the Model-Integrated Parallel Application Synthesizer is the only method supported directly by the MGA: declarative modeling. Generative modeling plays a secondary role. It is reflected by the fact that generative modeling is implemented through textual attributes which are assigned to graphical model components. A totally new model component cannot be generated, only replications of an existing graphical model can be. This is intentional; declarative modeling is a powerful methodology and its usage is favored in the MIPAS. Generative modeling is supported only to augment the capabilities of graphical model building. Its main purpose is the compact description of repetitive structures and the flexible specification of conditional components.

This double paradigm, declarative and generative, has a minor drawback. Neither the graphical, nor the textual model representation contains all the information about the system. Consequently, the models are hard to comprehend by humans. The MIPAS contains a special model interpreter dedicated to overcome this problem. The Model Transformation Tool (MTT) converts these mixed models to purely graphical representation by evaluating the generative attributes and creating a new model database. The MTT is described in greater detail in the following section.

Generative modeling is very useful for reducing complexity and speeding up the modeling process. Along with the MTT, it transforms modeling into a two-stage process. The user first creates the models with the mixed declarative and generative specifications. Then the MTT is used to transform the models. Then the user can apply the Graphical Model Builder again to check the models visually. While it is possible to modify the automatically generated models, it is not considered to be a good modeling practice, since the automatic transformation works in one direction only. There is no support provided to modify the original models automatically based on the manual changes of the models generated by the MTT.

4.2. Modeling Aspects

The objective of this research is to automatically synthesize large-scale instrumentation systems running on distributed memory multiprocessors with flexible interconnection topology. What do the system models need to contain to achieve this goal? There are two main aspects of the problem: the software and the hardware, i.e. the signal processing and other computations that need to be performed and the target hardware architecture. Consequently, one modeling aspect is assigned to each.

The signal flow graph is a widely accepted way of describing instrumentation/signal processing systems. The first modeling aspect, the Signal Flow Aspect, closely resembles a signal flow graph. The Hardware Aspect describes the available hardware resources and their interconnection topology. These two aspects of the system are not independent. Different elements of the signal flow graph may have certain hardware resource requirements. They constitute assignment constraints that must be satisfied during system synthesis. The third and final aspect of the system models describe these resource requirements. It is called the Assignment Constraints Aspect.

4.2.1. Signal Flow Aspect. The signal flow models consist of predefined atomic and user-defined aggregate components. The primitive and the compound are the aggregate model components of the Signal Flow Aspect.

The primitive is the lowest level computational block. It contains atomic objects only. It does not have any connections. It has different textual and numerical attributes associated with it, such as a script, a priority, etc. In addition to atomic objects, the compound contains user-defined components, i.e. previously defined primitives and compounds. Compounds containing compounds create the model hierarchy.

The signal flow model of a system must have exactly one top level compound model containing every lower level model. This hierarchy is best described by a tree. The nodes of this tree are the compound and primitive models. The children of a node are the aggregate models it contains. The root of the tree is the top level compound model. The leaves are the primitive models.

Primitive models correspond to actornodes in the Multigraph Computational Model (MCM) [4]. The most important attribute a primitive model has is the script. The script is a subroutine written in a procedural or functional language that is executed every time the actornode is fired. The script attribute contains the name of the subroutine and the object or library file name where it is located. A related attribute is the estimated execution time. This is needed by the system synthesizer to ensure good processor allocation.

The atomic objects that primitives can contain are the input and output signals, and the input and local parameters. The input and output signals constitute the data interface of the primitive. They correspond to actornode ports in the MCM. The Multigraph Kernel (MGK) provides a set of functions to access these "data ports" to receive or propagate data at runtime. At higher levels of the model hierarchy, the icons corresponding to the input and output signals are connected to create the signal flow graph. The attributes of input and output

signals include data rates. These are needed by the system synthesizer to ensure good allocation of communication resources.

The local and input parameters of the primitive model are used to assemble the actornode context. Local parameters have data types and values specified by the user. Input parameters are used to propagate the value of a local parameter specified at a higher level of the model hierarchy down to the primitive model. The local and input parameters can have simple data types, e.g. integers or doubles, or pointers to more complicated, user-defined types.

Compounds may contain primitives, compounds, and atomic objects. These atomic objects can be input, output and local signals, input and local parameters, and conditions. Local signals correspond to datanodes in the MCM. Their attributes include data type and buffer length. Input and output signals describe the data interface of the compound model. Icons representing primitive and compound components have ports for their input and output signals (and for their input parameters). The signal flow is modeled by connections between selected types of atomic components and ports of aggregate components (primitives and compounds). These connection constraints ensure that the resulting dataflow graph is bipartite as required by the MCM: actornodes are connected to datanodes and vice versa.

The value of each local parameter needs to be propagated down all the way to the primitive, where it becomes a parameter for the script as part of its context. This is modeled by connections between local and input parameters. As an example, consider a primitive model corresponding to a simple amplifier actornode. It has one input signal for the input data, one input parameter for the gain, and one output signal for the output data. Several instances of this model can be used in different compound models. For each instance, a local parameter needs to be defined and connected to the input parameter of the amplifier primitive. The value of this local parameter is the gain, which can be different for each instance of the amplifier.

Generative modeling is supported through generative attributes in the MIPAS. In the signal flow aspect, every primitive, compound, input, output and local signal, and input and local parameter has a repetition textual attribute. This is specified as a C++ function body that returns an integer, the number of repetitions of the model component. The repetition generative attribute defaults to "return 1;".

Model connections can be specified in the structure attribute. Only compound models have inner structure, therefore, only they have this attribute. The structure attribute is specified as a C++ function body. In the code, components of the current model can be accessed by name. Connections can be created or destroyed by calling the predefined functions Connect(a,b) and Disconnect(a,b). These are overloaded C++ functions. They allow parameter type combinations for all legal connection types. For example, input signal to input signal port, output signal port to local signal etc.

Compound models can contain one or more conditions. Conditions are atomic objects. They contain a user specified numerical value. This value can be accessed by name in the repetition and structure attributes. Generative modeling and conditions provide a very flexible and powerful modeling technique.

4.2.2. Hardware Aspect. The MIPAS target hardware platforms are distributed memory multiprocessors with flexible interconnection topology, such as TMS320C40 or SHARC networks. The key information the models need to capture are the topology of the network and the available resources.

The hierarchy of the hardware aspect is organized in a manner similar to that of the signal flow. The concept of the two user-defined components, the node and the network, is similar to that of the primitive and the compound.

A node can have only atomic parts, e.g. communication links, while a network can have node and network components as well. Node models correspond to processors, while network models describe uni- or multiprocessor boards, subsystems, systems, etc.

The communication links of nodes have maximum data rate attributes specifying their speed. The nodes have attributes specifying their performance. These are needed by the system synthesizer for resource allocation.

Networks can have node, network, as well as communication link components. Nodes and networks can have resources attached to them. Resources are atomic objects. Their only attributes are their names. They represent special capabilities of nodes or networks, for example, such devices as A/D converters, disks, or printers attached to them. They can be used in the assignment constraints aspect to express resource requirements of different blocks of the signal flow model.

Generative modeling in the hardware aspect is similar to that in the signal flow aspect. Nodes, networks, and ports have the repetition attribute. Networks have the structure attribute as well.

4.2.3. Assignment Constraints Aspect. The Assignment Constraints Aspect constitutes a two-level hierarchy. The top level model, called configuration, contains a set of lower level models, called rules and the bans. Rules specify positive assignment constraints, e.g. that a given signal flow module must be assigned to a given hardware module. Bans specify negative assignment constraints, e.g. that a given signal flow module must not be assigned to a given hardware module.

Rules and bans can have references to user-defined signal flow model components and to user-defined hardware model components. Rules can have resource requirement parts. Resource requirements are atomic objects, they represent special needs of the signal flow components. A model component can be referenced as a type or as an instance. Assigning a specific computation to a specific processor requires instance references. But assigning a type of computation, e.g. a generic FFT primitive, to a class of nodes, e.g. digital signal processors, requires type references.

A rule must contain one or more signal flow references, and can contain one or more hardware references. The signal flow references of a rule mean that the run-time objects corresponding to the (type of) signal flow components must be assigned to the same node. If the rule contains a hardware reference that means that the run-time objects must be assigned to one of the processors specified by the reference. If there are more then one hardware references, then the assignment can be made to any one of them. If the rule contains resources requirements, that means that the assignment must be made to a node containing all the specified resources or to a node specified by a hardware reference.

These relations can be expressed by the logical expression:

$$[SF_1 \wedge ... \wedge SF_n] \rightarrow [HW_1 \vee ... \vee HW_m \vee HWnode\{RS_1 \wedge ... \wedge RS_l\}]$$

meaning that run-time objects corresponding to SF_i signal flow component references (type or instance) must be assigned together to one of the processors corresponding to HW_j hardware model component references (type or instance) or any of the processors having every RS_k resource. The hardware and/or the resource component part can be omitted, but at least one signal flow component must be present.

A ban must contain one or more signal flow references, and one or more hardware references. The signal flow references of a rule mean that none of the run-time objects corresponding to the (type of) signal flow components may be assigned to any of the processors specified by the hardware references.

These relations can be expressed by the logical expression:

$$\sim\{[SF_1 \vee ... \vee SF_n] \rightarrow [HW_1 \vee ... \vee HW_m]\}$$

meaning that none of the run-time objects corresponding to SF_i signal flow component references (type or instance) may be assigned to any of the processors corresponding to HW_j hardware model component references (type or instance).

Every rule and ban has a reference generative attribute. It is used to reference generative model components of other aspects. The reference attribute specializes the meaning of the graphically generated references. It can either modify the current rule (ban) or create a new one.

The reference attribute contains a C++ function body. The predefined function Refer(ref,i) is used to specify the instance of a model the graphical reference points to. For instance, if a rule contains a reference FFT to a signal flow compound FFT and the repetition attribute of this model specifies 5 instances, then the reference attribute Refer(FFT,3) specifies that the FFT reference points to the fourth instance of the FFT compound. A single reference can have several instantiations by multiple Refer(ref,i) calls.

5. Model Interpretation

The task of the model interpretation in the Multigraph Architecture is to synthesize applications from the domain models and run-time libraries, and to produce input to various system engineering tools. In the case of the parallel signal processing domain, there are two different model interpreters and a model analysis tool responsible for distinct tasks. Figure 2 illustrates their location in the overall structure of the Model-Integrated Parallel Application Synthesizer (MIPAS).

5.1. The Model Transformation Tool

The task of the Model Transformation Tool (MTT) is to visualize generative models to help debug system models. The models in the MIPAS are mixed declarative (graphical) and generative (textual). It is relatively easy to make a mistake, which can go undetected in the modeling phase, because of this double paradigm. The MTT takes the system models, evaluates the generative attributes of the model components, and generates a new set of models that are purely declarative (graphical) and, therefore, easier to debug.

The MTT has two main parts. The first one evaluates the models and creates the second part, a C++ program, automatically. It generates data structures based on the models, and writes wrappers around the function bodies specified by the user as generative attributes. The generated program contains the predefined functions for each aspect and a main function. The main function creates the appropriate number of replications for each model component by calling the functions created from the repetition attributes, makes the additional connections using the functions generated from the structure attributes, and creates the references specified by the reference attributes. This second part lays out the model components and routes the connections automatically for the GUI of the model builder. The generated C++ program is compiled, linked, and executed. Syntax errors

in the user-specified attributes cause compiler or linker errors. Since these errors are not easy to trace back to the models, a tool is needed to locate the original errors in the model specifications automatically.

5.2. The Parallel Application Builder

The primary model interpreter in the MIPAS is the Parallel Application Builder (PAB). It is responsible for: (1) creating the macro dataflow graph corresponding to the signal flow models, (2) partitioning the graph, (3) assigning the partitions to the nodes of the processor network specified in the hardware models while satisfying the requirements specified in the assignment constraint models, (4) creating the executables for the nodes, and (5) providing the hardware description and communication information to the Graphical Configuration Manager (GCM), the model analysis tool responsible for hardware diagnostics and model verification, message routing, and network loader configuration (Figure 2).

The PAB first creates a signal flow builder object network corresponding to the signal flow models residing in the model database. There is a builder object corresponding to every model object, including compounds, primitives, signals, parameters, and conditions. The generative attributes are evaluated to create the currently required number of builder objects. Either the original models, or the models generated by the Model Transformation Tool (MTT) can be provided to the PAB. The resulting builder object network is a tree, the root of which is the top level model builder corresponding to the top level signal flow model.

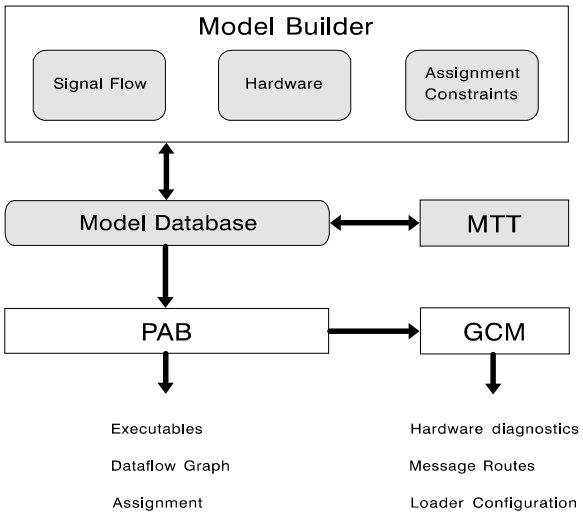

Figure 2. The MIPAS Architecture

The PAB creates the connections specified in the generative attributes of the models. The program checks every model connection (signal and parameter) for datatype consistency and creates the appropriate connections in the builder network as well. All direct connections (between primitive input and output signals and local signals, and between local parameters and primitive input parameters) corresponding to the connections in the dataflow graph are generated bypassing the hierarchy. Next the corresponding MCM objects are created: actornodes for primitives, datanodes for direct signal connections (or local signals), and contexts for parameters. Then the actornodes and datanodes are connected to form the dataflow graph.

The next step is to make a builder network for the hardware models. This is performed similarly to the signal flow model interpretation: a builder tree is created, where the nodes are the network, node, communication link, and resource component builders. All the connections between model components and the direct node to node connections are generated as well.

The PAB evaluates the rules and bans of the assignment constraints configuration and provides each signal flow primitive builder with a list of hardware node builders the corresponding actor can be assigned to. Infeasible requirements are detected at this point. Since the assignment problem is NP-complete, the PAB employs a heuristic procedure to partition the dataflow graph and assign the subgraphs to the hardware nodes. It utilizes the hierarchy of the signal flow models to cut the search space and guide the search. The approach to the assignment problem is beyond the scope of this paper. It is described in detail in [5].

An alternative to the traditional assignment procedure, where one assigns processes to a preconfigured hardware platform, is hardware topology synthesis. The input to this procedure is the signal flow graph and a set of nodes. Hardware topology synthesis creates the processor interconnection network. It tries to match the topology of it to that of the signal flow graph. This problem is NP-complete as well, therefore, a heuristic approach must be used.

The PAB implements hardware topology synthesis utilizing the hierarchy of the signal flow models as a user-defined heuristic. The approach is described in [5].

Once the assignment is completed, the PAB generates a makefile to link the appropriate MGK and the required object and library files to create the executable for each processor. Then the make utility is executed.

Before activating the dataflow graph and starting the execution of the application, message route generation and network loader configuration needs to be performed. These tasks are carried out by a model analysis tool, the Graphical Configuration Manager (GCM).

The PAB generates input information required by the GCM. The program is capable of comparing the actual hardware configuration to the hardware models, generating network loader configuration files and

deadlock-free message routing for wormhole and store-and-forward routers [5-6].

The purpose of comparing the hardware models to the actual processor network is twofold. First, it validates the models and second, it diagnoses the hardware itself. Another important task of the GCM is network loader configuration. The program is capable of generating configuration files for different loaders. The most important task of the GCM is deadlock-free message route generation.

Deadlock avoidance with store-and-forward and virtual cut-through routing is simple with a careful message buffer allocation strategy. Wormhole routing, the most efficient message routing method, is, however, deadlock-prone. The two known deadlock avoidance methods cannot be utilized. Virtual channels require hardware support not available on the MIPAS target platforms. Topology-based deadlock avoidance is too restrictive for the purposes of the MIPAS.

Partially connected message routing can be used in the MIPAS because the PAB provides not only the description of the hardware, but also the list of processors that need to communicate with each other. In [5], it is shown that partially connected minimal deadlock-free routing is NP-complete. Consequently, a non-minimal approach must be used. A partially connected, non-minimal message routing strategy, that guarantees deadlock-freedom and provides comparable, in many cases smaller, communication overhead than minimal routing strategies, is introduced and evaluated in [5]. The GCM incorporates this algorithm to generate deadlock-free message routing for wormhole routed networks.

6. Conclusions

This paper has discussed the adoption of a model-integrated programming environment, the Multigraph Architecture, to the parallel instrumentation and signal processing domain. The target hardware architecture is distributed memory multiprocessors with flexible interconnection topology. The declarative modeling methodology of the Multigraph Architecture (MGA) has been extended by generative capabilities providing a powerful new paradigm. This research has solved the previously open problem of deadlock-free wormhole routing in networks with arbitrary topologies. Process assignment has been solved automatically by a non-optimal, heuristic search procedure and by hardware topology synthesis. Both algorithms utilize the hierarchical structure of the system models providing a user-defined, application-specific heuristic.

7. References

[1] Sztipanovits, J., "MULTIGRAPH: An Architecture for Model-Integrated Computing," *Proceedings of the International. Conference on Engineering o Complex Computer Systems*, Ft. Lauderdale, FL, November 1995

[2] Karsai, G., "A Visual Programming Environment for Domain-Specific Model-Based Programming," *IEEE Computer*, March 1995

[3] Perry, D., *VHDL*, McGraw-Hill, 1991

[4] Abbott, B. A., et al., "Model-Based Software Synthesis," *IEEE Software*, May 1993

[5] Ledeczi, A., "Parallel Systems with Flexible Topology," Ph.D. Dissertation, Dept. of Electrical and Computer Engineering, Vanderbilt University, December 1995

[6] Ledeczi, A., Abbott, B. A., "Parallel Systems with Flexible Topology," *Proceedings of the Scalable High-Performance Computing Conference*, Knoxville, TN, May 1994