AIRES TOOLKIT USER MANUAL

For MOBIES

Version 1.0

Released Date: December 2001

AIRES Group

Real-Time Computing Laboratory

EECS

The University of Michigan

1 Introduction

AIRES toolkit is a set of timing analysis and design view construction algorithms integrated in the GME environment. The objective of developing this toolkit is to support automatic transformation of design views to runtime views with the consideration of non-functional constraints, and analysis and verification of system timing properties. The toolkit is also used as a means of evaluating a set of algorithms for view transformation, timing assignment and analysis, which we have developed in the DARPA/ITO MoBIES project context. Although these algorithms are integrated in the GME environment, they are all implemented as plug-ins, and should also be integratable in other modeling environments.

In the current version, the view transformation and timing analysis algorithms are all invoked through GME interpreter mechanism. Given an application model constructed in GME using the predefined meta-model, the interpreter will call the plug-in algorithms to analyze the application model. The following features are supported in current AIRES interpreter:

- Allocating software components in design model to a set of pre-defined tasks;
- Distributing end-to-end timing constraints over a set inter-communicating tasks;
- Assigning tasks to a set of pre-defined processors with tasks' timing properties;
- Schedulability analysis with a generalized RMA;
- Code generation for single processor platform.

To create application models for timing, allocation and schedulability analysis in the GME environment, a meta-model is required. We released a meta-model created by ourselves for this purpose (together with this release, and packed in the zip file). It is intended to be used for both automotive and avionics OEPs. It is different from the Analysis Interchange Format (AIF) defined by the avionics OEP. Certainly users can create their own meta-model for a target domain, but changing meta-models in GME requires rewriting the interpreter to convert the information in GME to formats as required by our algorithms

This release contains AIRES meta-model, AIRES interpreter, and a sample application model based on ETC developed using the AIRES meta-model.

2 Installation

Current AIRES toolkit can only run on machines with Microsoft Windows operating systems due to the availability of GME. Machines with NT 4.0 and 2000 should have latest service pack installed.

Installation of AIRES toolkit requires about 20 MB free disk space, and GME 2000 pre-installed on the host.

2.1 Install GME 2000

Download GME 2000 from Vanderbilt University website at:
http://www.isis.vanderbilt.edu/
For instructions of installing GME, refer to GME user manual.

2.2 Install AIRES meta-model

Download the AIRES toolkit from our project website at:
http://kabru.eecs.umich.edu/aires
Unzip all the files in the package to a directory (e.g., C:\Aires). All the following components have to be installed for the correct functioning of the AIRES interpreter:

- AIRES meta-model, defined in file “mobies.xml”;
- Component-to-task mapping algorithm;
- Deadline distribution and timing property assignment algorithm;
- Task allocation algorithm;
- Scheduling analysis algorithm.

Follow steps below to import the AIRES meta-model: Start GME v1.2

- Click File->New->Add from File
- Open "mobies.xml" in the AIRES directory. A paradigm named “mobies” should appear after this step, as shown in Figure 1.
- Click Close

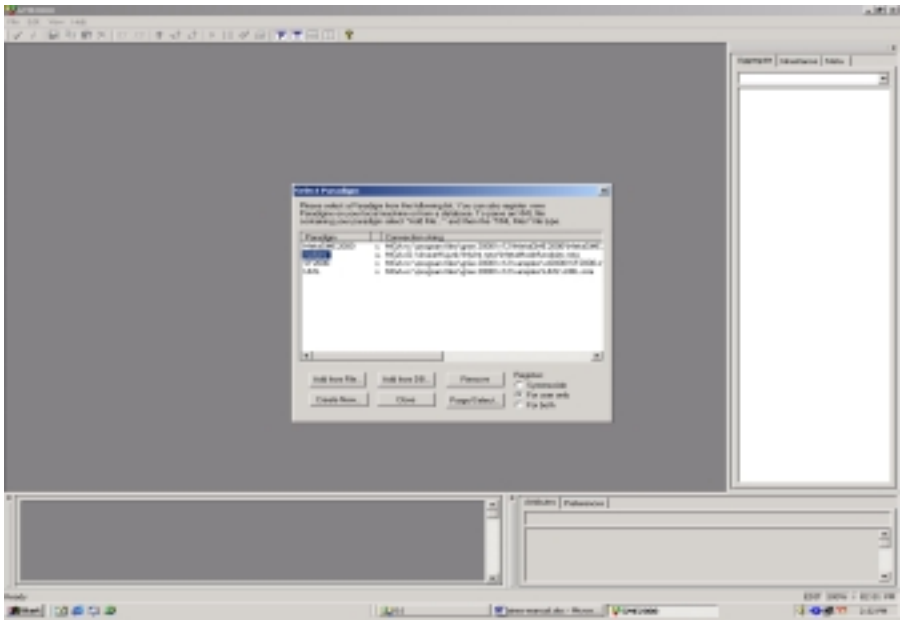

Figure 1

(Note that the user can construct different meta-model as described in GME user manual. However, to make the plug-in AIRES algorithms working properly, the user has to define the required timing information used by AIRES algorithms. Otherwise, the interpreter has to be rewritten to translate the information in GUI to the formats used by the algorithm.)

2.3 Build the AIRES interpreter

Open the MSVC project "library.dsw" in the folder "MoBIES_MI/library" and build "mobies.lib". Open the MSVC project "BONComponent.dsw" in the folder "MoBIES_MI/Interpreter", and build it. Make sure that "mobies.lib" is in the path for linking in libraries. If successful, it will give a message "Performing registration RegSvr32: DllRegisterServer in .\Debug\Component.dll succeeded.", which means that the AIRES interpreter has been registered in the Windows registry. An alternative way to install the AIRES interpreter is described below.

2.4 Install AIRES interpreter

All AIRES plug-in algorithms have to be invoked using GME interpreter mechanism. The plug-in algorithms have been packed as a dynamic link library (.dll file for Windows). They need to be installed in the Windows Registry to be invoked by the interpreter of GME and function correctly at runtime. To install interpreter for AIRES analysis algorithms, execute the following steps:

- Create a new project in GME, for example, mobies.
- Click File->Register Components->Install New
- Open "Component.dll" in the AIRES directory. If installed correctly, the new interpreter will be shown as Figure 4 in the component window.
- Click Close

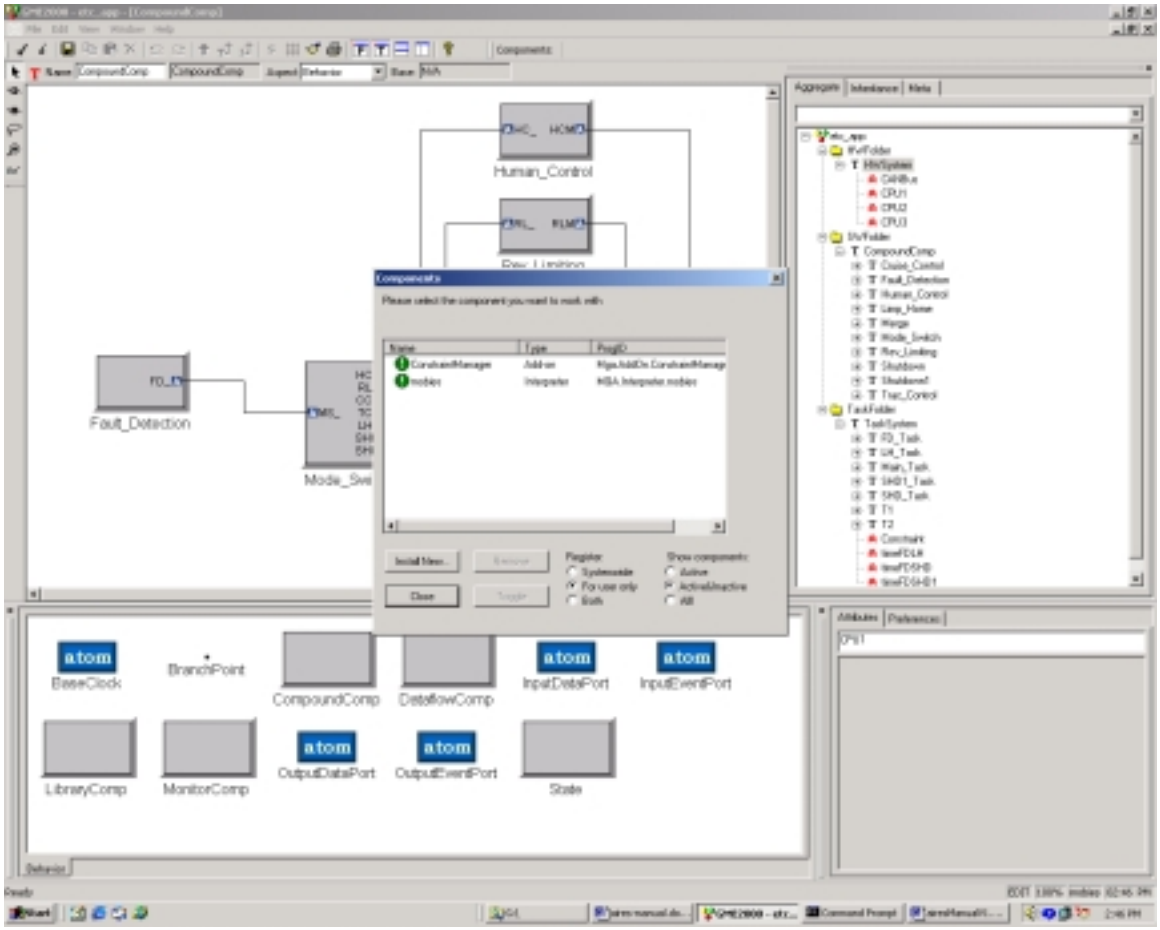

Figure 2

2.5 Install required software packages

Automatic code generation in AIRES requires several packages from different sources. These packages are optional if code generation is not what is interested.

2.5.1 Graphviz

Graphviz is used to visualize the results of component mapping to tasks. It can be downloaded from:

http://www.research.att.com/sw/tools/graphviz/

If this tool is not installed, the result of component-to-task mapping will be only given in text format.

2.5.2 TMAKE

The TMAKE program is used for automatic code generation. TMAKE is not required if there will not be any automatic code generation. To install TMAKE, download TMAKE package from:

http://www.trolltech.com/developer/download/tmake.html

and install it on the host machine.

3 Timing Analysis With AIRES ToolKit

3.1 Basic Concept

AIRES toolkit allows designers to specify the software structure (design architecture), runtime structure, physical platform configuration and timing constraints. Timing constraints specified in MoBIES meta-model include end-to-end deadlines, rate constraints and worst-case execution times for each task in a runtime structure. In this documentation, we only discuss the elements related to timing analysis in AIRES toolkit. For detailed information on creating models, specifying attributes, etc., please refer to related GME documentation.

Components and design architecture. Components are basic building blocks in AIRES model. Each component has a set of input ports and output ports associated with it as interfaces to interact with other components in integration. In current AIRES meta-model, there are 2 types of components and 2 types of ports are defined:

- Dataflow components
- Composite components (called "CompoundComponent in the tool)
- Data ports
- Event ports

Given these components and ports, a software design model (or design architecture) can be constructed by instantiating components and connecting their ports.

Tasks, task graph and runtime architecture. Tasks are basic units for schedule. Each task can be implemented as a process or thread in operating system, and can have its own scheduling attributes such as priority, scheduling policy and period. This is different from the application-level task concept, which in some domain, indicates a whole process from sensing external status, to computing proper commands and sending them out. In our terminology, such application-level tasks are called *system threads*. A system thread can be implemented as one or more operating system threads/processes (called tasks) with each does part of the job such sensing data, computation and command output. A task can also implement multiple system threads in it. System threads are normally specified in design models, while tasks are usually in runtime models.

A task group consists of a set of tasks and their interactions. Different from the interactions in design model, the task interactions are normally implemented using inter-process communications (IPC), which require the underlying system services support and consume system resources as well. Although the task graph itself can be used to describe invocation dependency and data dependency, we use it for data dependency in our AIRES toolkit for now. In current AIRES meta-model, each task has the following properties:

- Task name
- Input ports and output ports
- WCET
- Rates

Given a set of tasks and their interactions, the runtime architecture can be specified as a task graph.

Constraints. Constraints are conditions that the system has to satisfy to ensure correct behaviors. In AIRES toolkit, we deal with 2 types of timing constraints: end-to-end deadlines and rates. End-to-end deadlines are the timing constraints applied to system threads, which indicate the time within which the operation has to be finished. Such an operation may have multiple steps in between, each of which may be implemented as a task.
Rate constraints are for individual tasks as well as system threads. Rate constraints specify how frequently a task/system thread will be executed.

Schedulability Analysis. Schedulability analysis is a process of verifying if all tasks can satisfy their timing constraints. If the task set is schedulable, it is safe to run them with assigned properties at runtime, and the system will behave correctly. Otherwise, the system may fail. The current AIRES toolkit uses a classical fixed-priority schedulability analysis algorithm, the generic rate-monotonic scheduling analysis, for scheduling analysis. This algorithm relaxes the assumption of RMA that relative deadlines are equal to tasks' periods and all tasks have to be independent. The following information are defined in AIRES meta-model for scheduling analysis:

- Task graph
- Task attributes, including WCET, rate, task release offsets and deadlines
- Inter-task communication cost
- Task allocations, i.e., on which processor each task will reside and execute

The result of the analysis will provide the information of:

- Whether the task set is schedulable (an Yes/No answer)
- Utilization of each processor

Platform configuration. Platform configuration defines the environment where the designed software will execute. The platform configuration includes hardware, operating system, middleware, and network in AIRES toolkit. The platform configuration can be specified in AIRES toolkit by selecting the corresponding processors and the network connecting them in current AIRES toolkit. The following platform properties are defined in the current AIRES meta-model:

3.2 Component Organization

All the models created using AIRES meta-model must have the following three folders as shown in Figure 3:

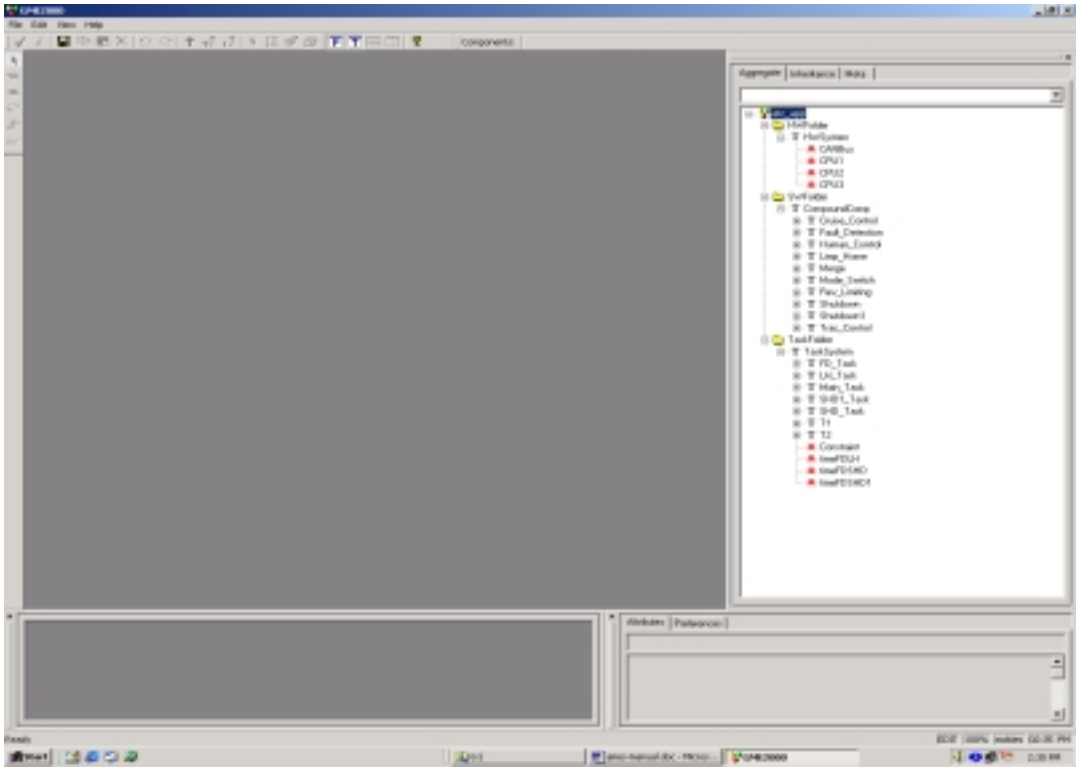

Figure 3

Hardware component folder: This folder contains a model of the hardware configuration, namely the processors, and the interconnection network. Currently the interpreter assumes all CPUs to be identical and connection is by a bus. The topology of the model is ignored while analyzing for task allocation and scheduling. Figure 4 shows an example hardware configuration.

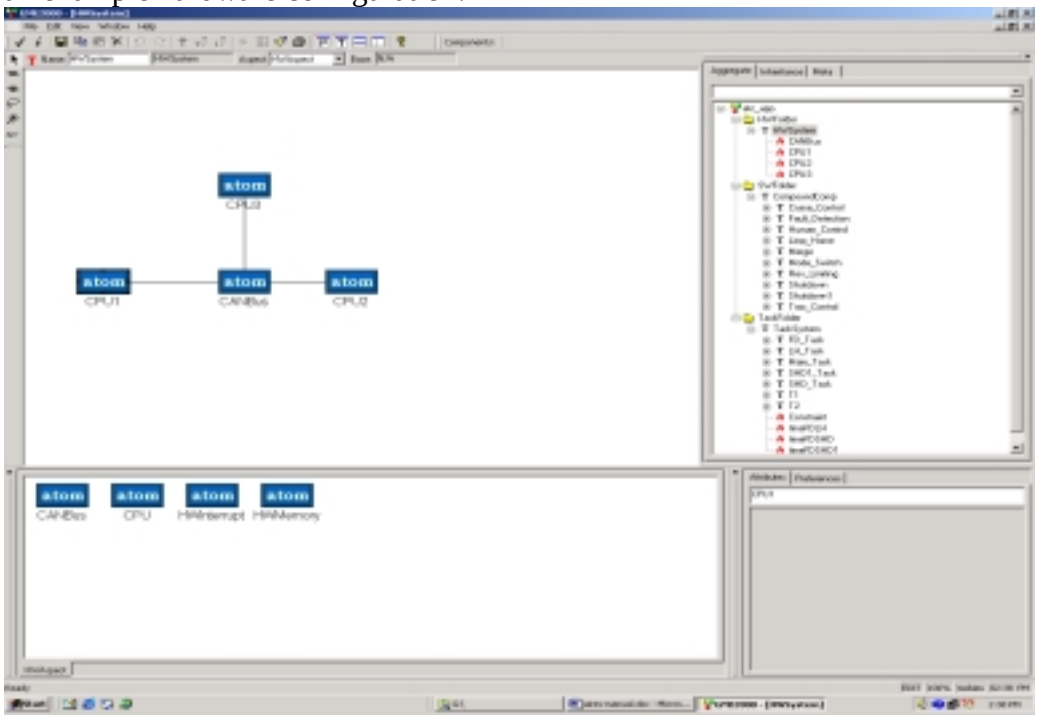

Figure 4.

Software component folder: This folder contains the software components or modules that make up the application. The software components can be interconnected to form a graph. Attributes like component execution time, communication cost between components can be specified. A software configuration is shown in Figure 5.

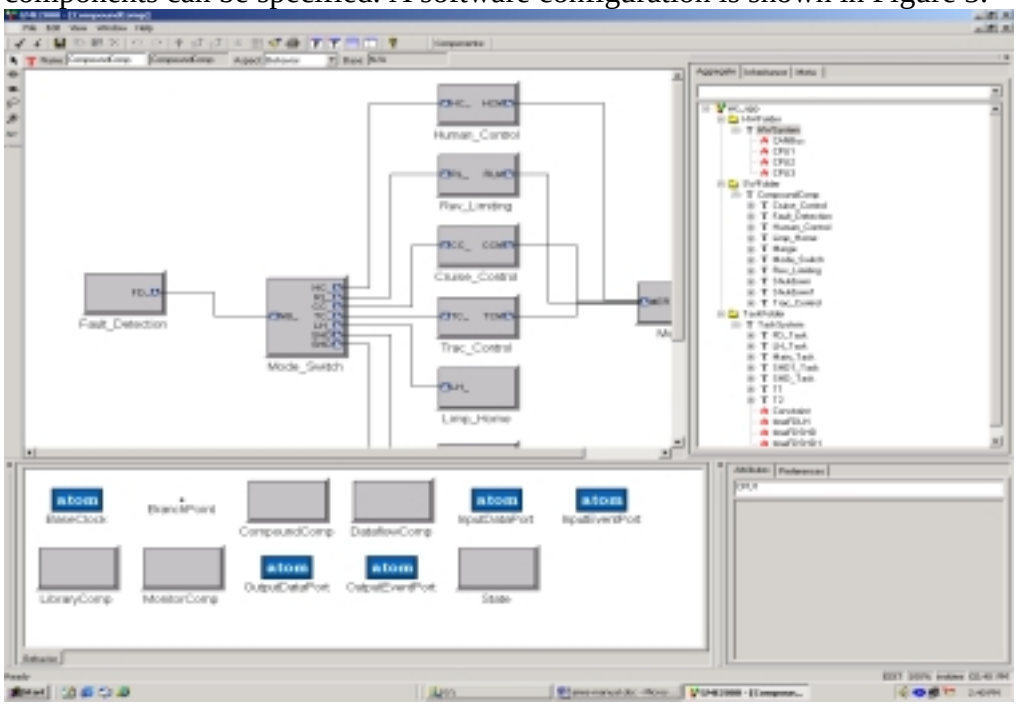

Figure 5.

Task folder: This folder contains the results of the component-to-task mapping. The results from the component to task mapping function of the interpreter have to be manually fed into the model. Currently an easy way to feed back results into the model is under developed by GME group. This folder contains a view of the application as a set of dependent tasks. The application model can be further enhanced by specifying the end-to-end timing constraints between tasks and the rates of tasks. A deadline constraint must be specified on each output task using the Constraint atom.

The execution time of the tasks and the inter-task communication cost are derived from the output of the component to task mapping function of the interpreter and is manually input to the model. Figure 6 shows an example task graph with timing constraints specified.

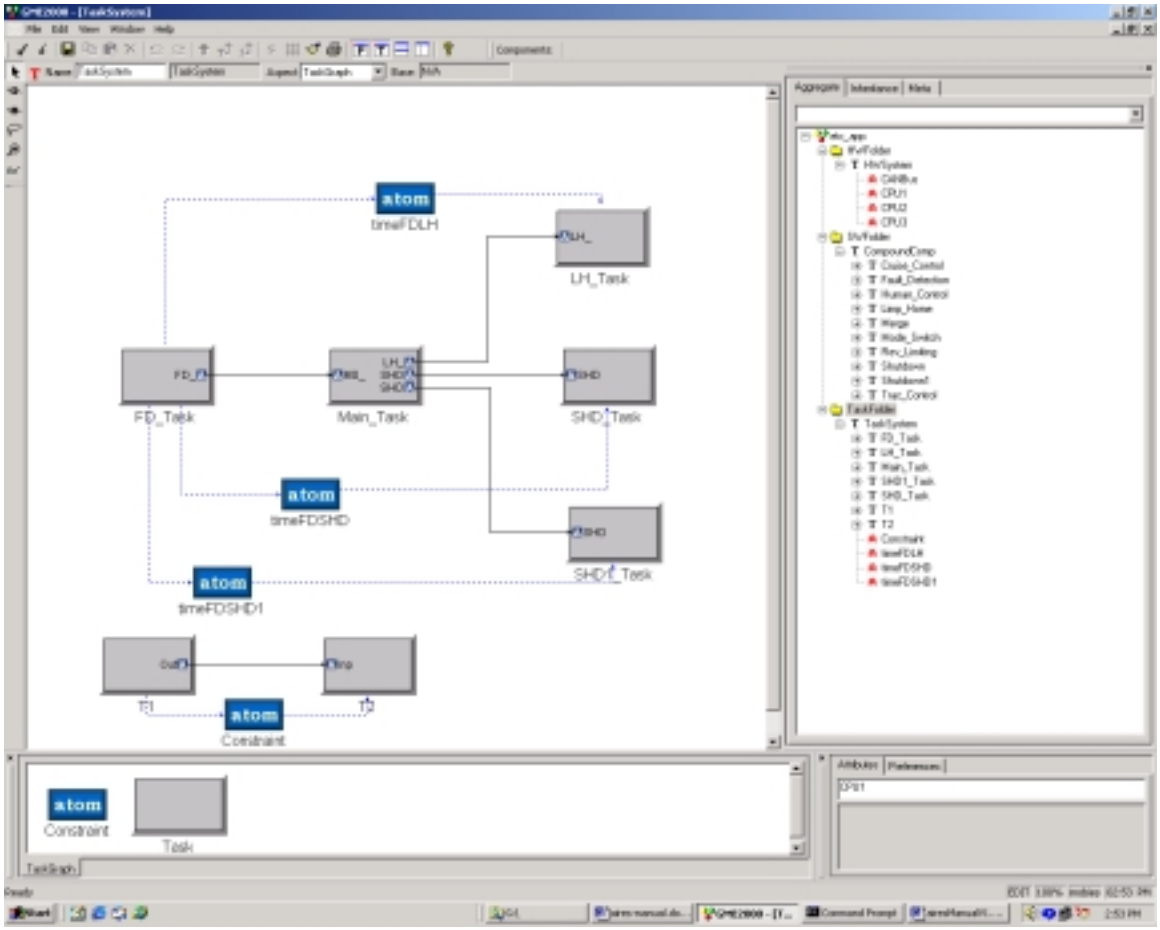

Figure 6

In the following subsections, we will use a model of the ETC application as an example to explain how different models can be constructed and how the timing analysis can be done step-by-step.

3.3 Design Model Construction

A design model can be constructed by selecting components from the meta-model element window, dragging and dropping them in the workspace, customizing them with properties (name, input and output ports, data size and frequency, etc.), and linking the ports of different components. The final software model should be stored in the software folder.

For the ETC example software, there are 10 components. The final design model is shown in Figure 7.

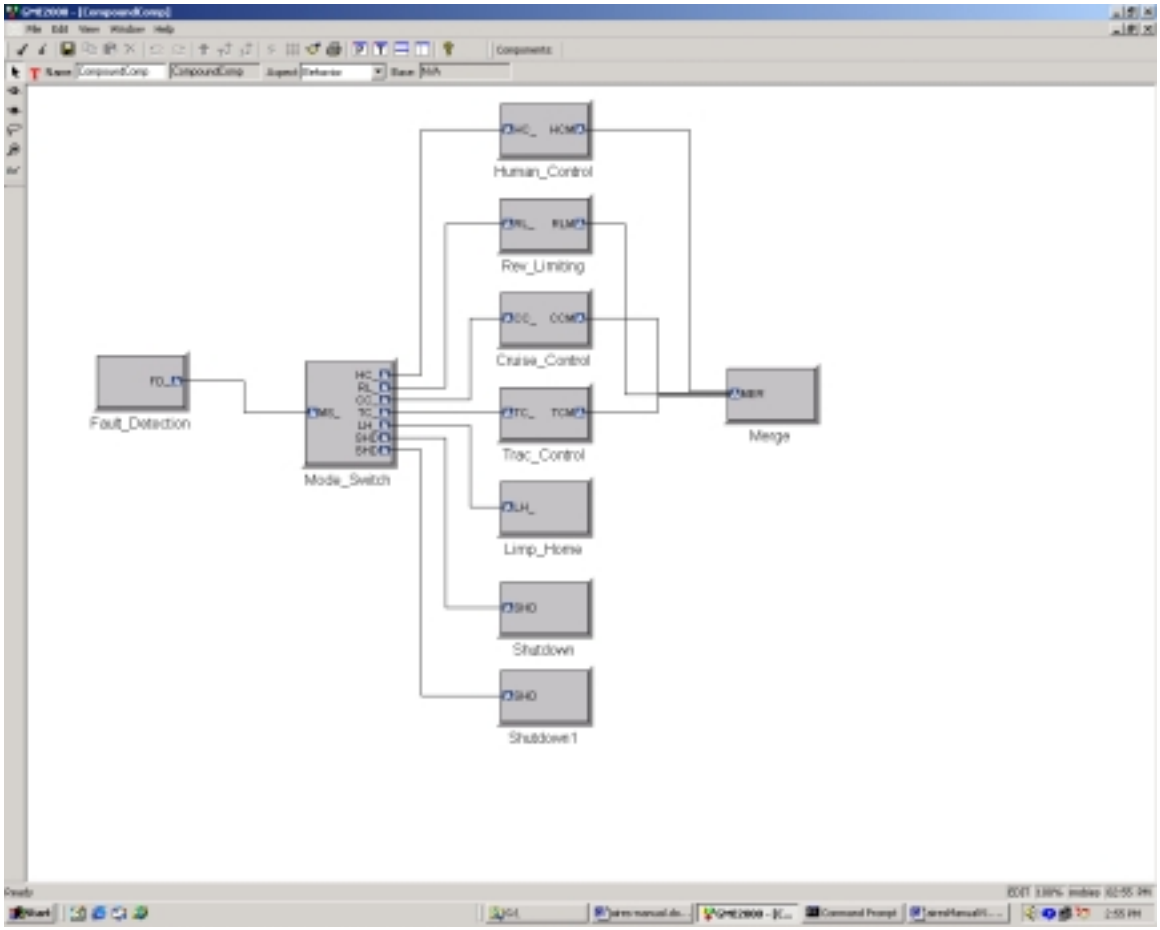

Figure 7.

3.4 Runtime Model Generation

Runtime model generation is an automatic process in AIRES toolkit. The generation is done through component-to-task mapping. The algorithm of component-to-task mapping is based on k-way cuts of a graph. In the current version of the algorithm, minimizing inter-component communication cost is used as a criteria to group components into tasks.

In this step, a component graph generated during the design model construction is used as input. The weight of each edge is then defined as the communication cost between two interactive components, which can be calculated using the data size and frequency passed along each link. The output of this algorithm is a task graph. The result will then be generated in both textual and graphic formats. The generated task graph is stored in Task Folder.

To invoke the component-to-task mapping algorithm, one must follow the steps below:

- Click the Task Folder in the Browser.
- Invoke the interpreter by either clicking on the interpret icon in the toolbar, or select the interpret option in the File menu. Note that the design model has to have already been constructed and stored in the SW folder.

- Click on Comp-Task Mapping button after the interpreter dialog box pops up.
- After the dialog box for the component-to-task mapping function pops up, specify the number of tasks intended to generate and the maximum number of components that a task can contain in the component-task mapping window. The option of maximum number of components in each task is intended to balance the load among tasks. Currently, this option outweighs the number of tasks, meaning that the algorithm will try to allocate maximum number of components in a task before moving to another task.
- Click 'Do Mapping' to perform component-to-task mapping.

The algorithm outputs both textual and graphic results. For the ETC example, the results are shown in Figures 8 and 9.

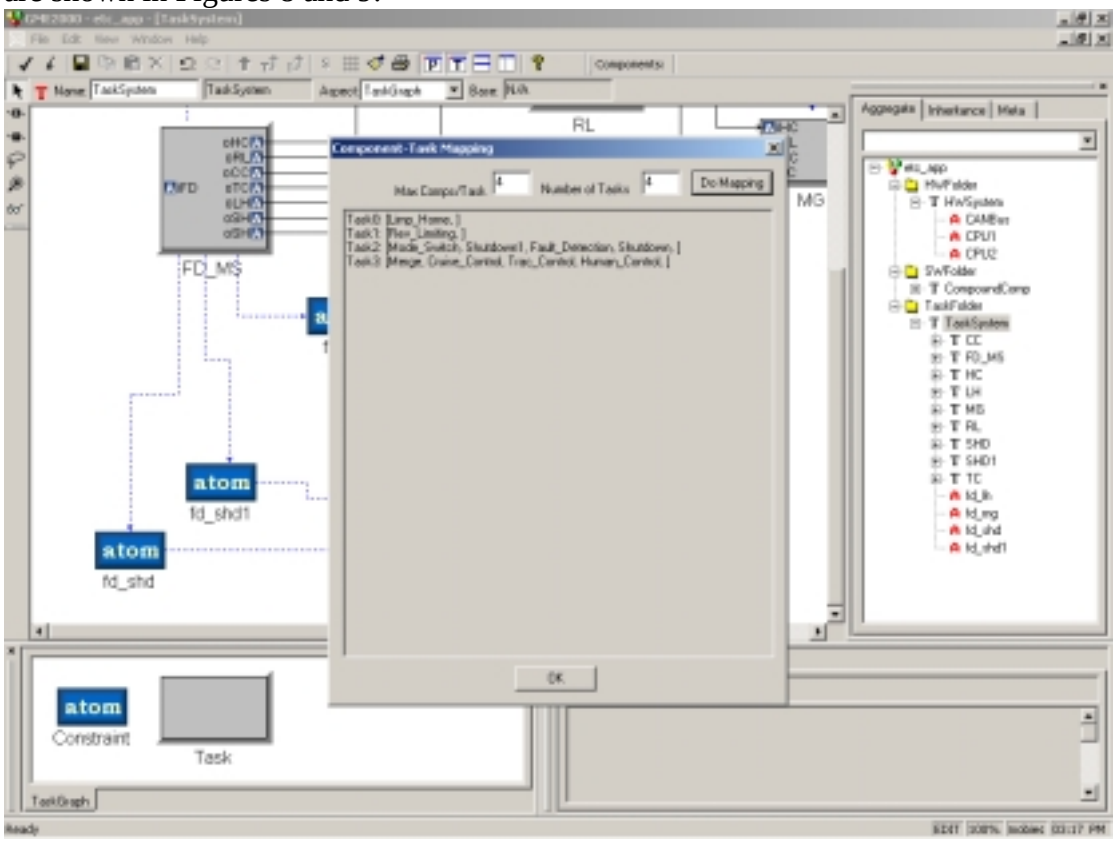

Figure 8

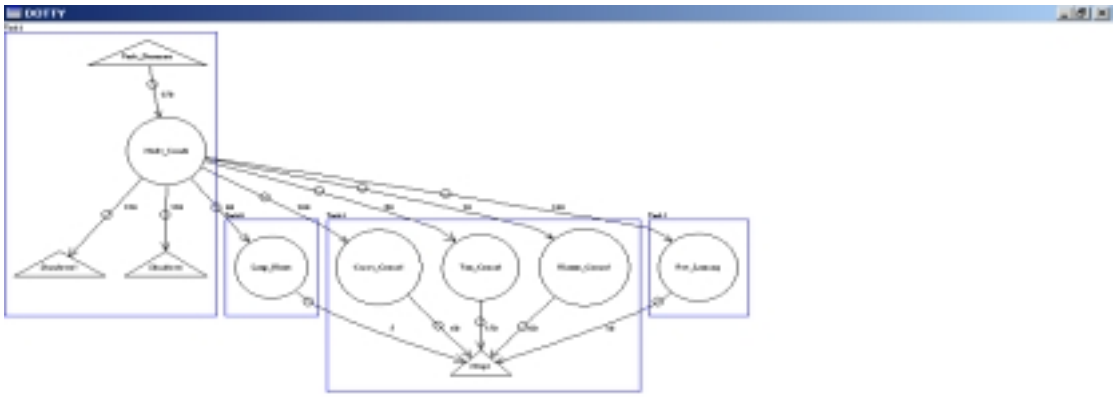

Figure 9

3.5 Deadline Distribution and Timing Assignments

Deadline distribution is a step to partition end-to-end timing constraints over a set of tasks with precedent constraints so that the overall timing constraints can be satisfied. The partitioned deadlines and the corresponding release offsets will then be assigned to tasks as their timing properties for scheduling and runtime control.

To do the deadline distribution and timing assignments, the following steps are required:

- Specify the timing constraints for the task graph in Task Folder by selecting constraint icons in meta-model window and linking them with tasks at the start and end. Then right-click the constraint icon to specify the value of each constraint. Task attributes such as rate constraints and worst-case execution time (WCET) can also be specified by right-clicking the task icon in the model and assigning values for each attribute. Multiple specifications are required if there are multiple constraints. For those tasks that are not subject to rate constraints, uncheck the value so that the timing assignment algorithm can assign proper values for those tasks automatically.
- Invoke the interpreter by clicking on the interpret option in the File menu.
- Click on RT Analysis button on the interpreter dialog box.
- Click on the Deadline Distribution button on the RT Analysis dialog box.

All tasks in the task graph are subject to some deadline constraints in order to perform the distribution and assignment correctly. If there exists some task that is with no constraint covers it, the interpreter will consider the specification as incomplete, and give an error message.

The results of deadline distribution are release offsets and deadlines for each task in the system. This can be used for sequencing and scheduling the tasks in the later stage.

Figure 10 shows the results of deadline distribution and timing assignment for ETC (constraints can be found in the example come with the package).

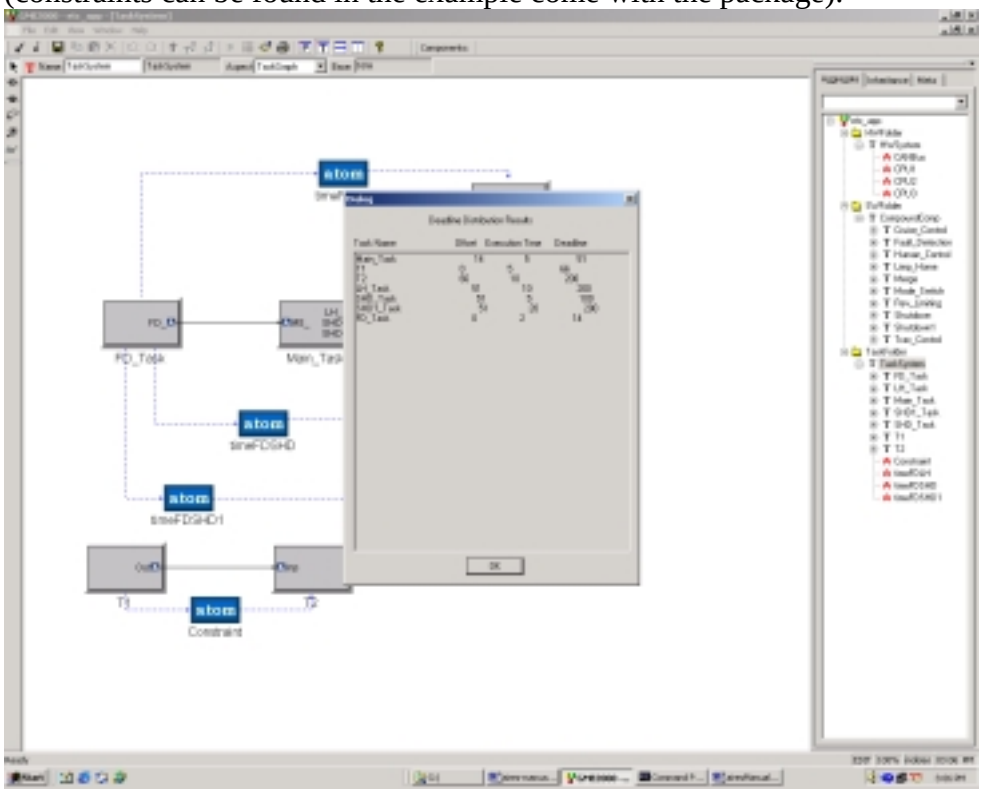

Figure 10.

3.6 Task Allocation and Schedulability Analysis

Task allocation is a process to assign tasks to different processors on a platform so that all constraints can be satisfied. Schedulability analysis then checks whether the set of tasks allocated to each process are schedulable. Current version of AIRES toolkit considers both processor utilization and communication cost between processors. The algorithm uses only first-fit for allocation, and generalized RMA for schedulability analysis.

Task allocation and scheduling analysis requires that the rate of each task be known prior to analysis. This can be ensured by specifying the period of each task in the model or by running deadline distribution prior to invoking Task Allocation and Scheduling. Correlation constant CF is used to control the way allocation clusters heavily communicating tasks together.

To do the task allocation and schedulability analysis, follow the steps below:

- Invoke the interpreter after the task graph has been constructed in Task folder and

deadline distribution and timing assignment have been done.

- Click on RT Analysis button on the pop-up dialog box.
- Click on Allocation+Scheduling button.
- If necessary, specify the correlation constant in the CF field of the Task Allocation and Scheduling dialog box.
- Click on Allocate+Schedule.

The results of task allocation and scheduling indicate if a successful allocation is possible by the algorithm. If successful, information regarding allocation of tasks to processors, utilization of each processor and task clustering are shown. The result for ETC example is shown in Figure 11.

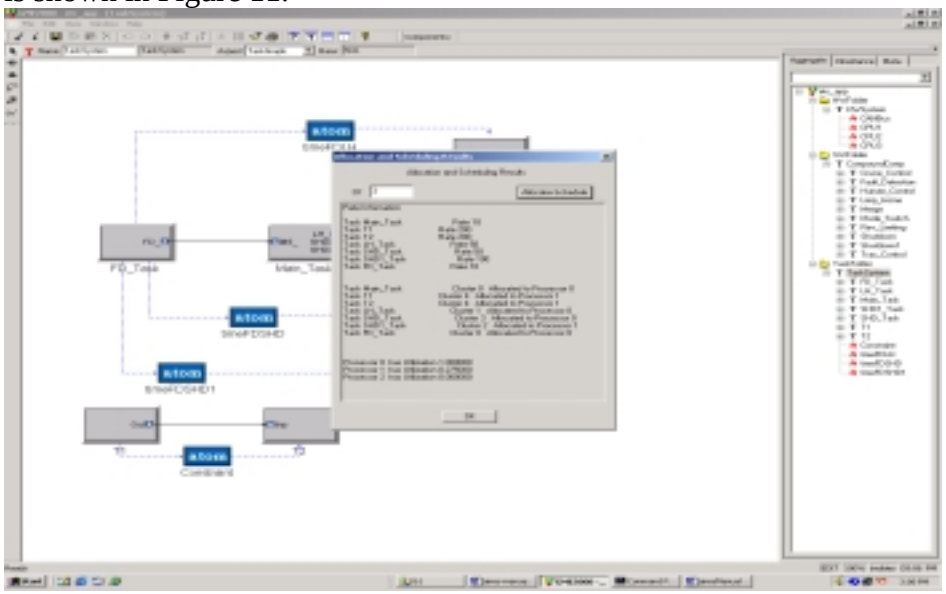

Figure 11

4 Code Generation

A simple code generator has been implemented at an early stage of the AIRES toolkit development, but efforts were discontinued as the MoBIES Phase I developers reached a clear partitioning of responsibilities for each contractor, and Michigan's focus has been determined to be in the real-time analysis area. However, the early code generation capability is still kept in the tool. As a result, the code generator is incomplete with several serious restrictions, such as:

- Only a single-processor platform is supported, as all inter-component communication is through shared variables.
- Only a single process/thread is generated that runs at a single rate.
- No performance optimization is attempted. Each block in the software component view translates into a C++ class in the generated code, unlike in tools like Simulink/Stateflow, where the entire diagram (with many blocks) is sometimes turned into a single function call. The generated code is a Microsoft Visual C++ project, which only runs on the Windows platform, not on the target

microprocessor. The intention is to enable running and testing the system's functional behavior, not the real-time behavior.

The code generator can be invoked either by invoking the interpreter from the toolbar, which allows one to generate code for the entire model, or by right-clicking on a component block, which allows one to either generate code for the entire model, or only for a specific block. Below we describe the steps for generating code for the entire model.

First, drag and drop a BaseClock block into the software component view, and specify its "period" attribute in milliseconds. This is the base rate that the system will run at. If no BaseClock is defined, then a default of 10 ms will be chosen.

At the initial interpreter screen, choose the radio button "Code Generation". In the next dialog box, choose "Full CodeGen", and specify a folder for the generated code. Note that this folder has to be an existing one, either empty or containing previously-generated code for the same model. If the folder does not exist, then one will be asked to create it before invoking the code generator. For testing purposes, create a folder at "c:\temp\test1", and put it in the dialog box. After clicking OK, the code is generated and another dialog box pops up, which allows you to view and edit the generated code. There are two panels, the left panel contains a list of generated files, one for each software component block containing a C++ class. In addition, there is a systeminit.cpp file that initializes the system block interconnection configuration. Clicking on the button "Regenerate Code" will regenerate all the code and overwrite your custom edits. Clicking on the button "Open MSVC" opens a MS Visual C++ project with the generator code skeletons, which you can fill in your custom, functional code, build and run the application. For example, you can put in a couple of printf statements in the execute() method of some of the classes and observe the system running and printing on a DOS window. Note that tmake and MSVC executables must be in the path for the "Open MSVC" button to work.

5 Contact Information

For further help or feedback, please contact any of the following persons:

- Kang G. Shin (kgshin@eecs.umich.edu)
- Sam Gu (zgu@eecs.umich.edu)
- Sharath Kodase (skodase@eecs.umich.edu)
- Shige Wang (wangsg@eecs.umich.edu)

6 Future Release

This is only the first version of the AIRES interpreter. Some of the algorithms used in

the analysis are simple. The meta-model doesn’t allow for overheads of the operating system to be specified. The interpreter doesn't do consistency checking between connected software components, or cyclic dependency checks. We plan to remedy this in the future version of the interpreter. We are also developing more advanced, scalable and efficient methods for allocation and schedulability analysis. These will also be incorporated in our future version. We are also planning to import from AIF to AIRES models and to export feedback information from AIRES models to AIF.