AIRES TOOLKIT USER MANUAL

Version 2.0

Released Date: February 2002

AIRES Group

Real-Time Computing Laboratory

EECS

The University of Michigan

What is new in version 2.0

New features added into this release include:

1. Upgraded the modeling environment from GME 1.2 to GME 2.0. New functionality supported by GME 2.0 can be found in related GME manuals.
2. Integrated component composition in the modeling environment. The function of component composition include importing Matlab Simulink/Stateflow diagram in .mdl files into GME environment, enabling to specify port information of each component for application construction, and checking the signal consistency of composed components.
3. Extended the software folder to include component models that can be used as a repository.
4. Extended the hardware folder to include operating system specification with both timer and scheduler overheads.
5. Separated interpreters for component to task mapping and schedulability analysis. The component to task mapping is used to automatically translate a design component model to a runtime task model, and schedulability analysis is used to generate runtime properties for tasks. These two functions are now implemented as separate interpreters triggered by different component icons in the tool bar. This modification is to enable schedulability analysis for those systems when manual translation of design model to runtime model is desired. In such a situation, a user can perform schedulability analysis directly without going through the steps for component to task mapping.
6. Task graph is extended to include the location of each task. This facilitates the changes across models of platform and analysis. By using platform reference in the task graph, the changes of platform such as processor type and OS overheads will be directly reflected in the task model, and will be used directly for analysis.
7. Added OIL file generation to facilitate the integration of analysis results and code generation for the system. The generated OIL file specifies only the task properties and supported objects (such as Alarm and Counter) for a single processor.

Use of the Fonts

We use different fonts to express different entities in the tool display.

Bold	name for a display window
Italic Bold	options in a menu or a list in a display
Italic	file names used by AIRES toolkit, including example files
Sans serif	file locations/directories

1 Introduction

AIRES toolkit consists of a set of plug-in algorithms for design model construction and reuse, runtime architecture generation and timing and schedulability analysis that are all integrated in the GME environment. The objective of developing this toolkit is to support reuse of design models and components, 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 the algorithms for composition checking, 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.

All functional components of AIRES toolkit, such as component composition, view transformation and timing analysis, are invoked through GME interpreter mechanism, with each is implemented as an individual interpreter associated to a meta-model. Given an application design 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:

- Importing Matlab Simulink/Stateflow diagrams into GME environment as reusable components;
- Modeling software system with reusable components, runtime system with a set of tasks, and platform with processors, network links and operating systems.
- Detecting any signal inconsistency in the software model;
- 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;
- Generating a configuration file with analysis results for each processor.

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 Automotive applications in this package. Users can create their own meta-model for a target domain, but changing meta-models in GME requires rewriting the interpreter to convert the modeling information in GME to formats required by our algorithms.

Although AIRES toolkit is designed and implemented targeting at automating the step-by-step design procedure starting from design model construction to runtime architecture construction and analysis to target configuration generation, the functionalities of AIRES toolkit can be used stand-alone. For example, one can use composition function in AIRES toolkit to reuse some Simulink block diagram to construct a software model, and feed it to some other tools for analysis. Similarly, one can also construct a task diagram from other modeling tools or manually, and perform timing and scheduling analysis using AIRES toolkit functions, then feed the analysis results to other code generation tool to generate code.

This release contains AIRES meta-model, interpreters for current AIRES functions, and an example 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 Obtain AIRES Toolkit

Download AIRES toolkit v2.0 from AIRES group website at:
http://kabru.eecs.umich.edu/aires
Note: This release of AIRES toolkit works only with GME v2.0. Previous release of AIRES toolkit v1.0 works only with GME v1.2, and does not have all functionalities described in this manual. Please refer to AIRES toolkit v1.0 manual for details.

Unzip all the files in the package to a directory (e.g., C:\Aires_Release_v2.0). The package should include the following components:

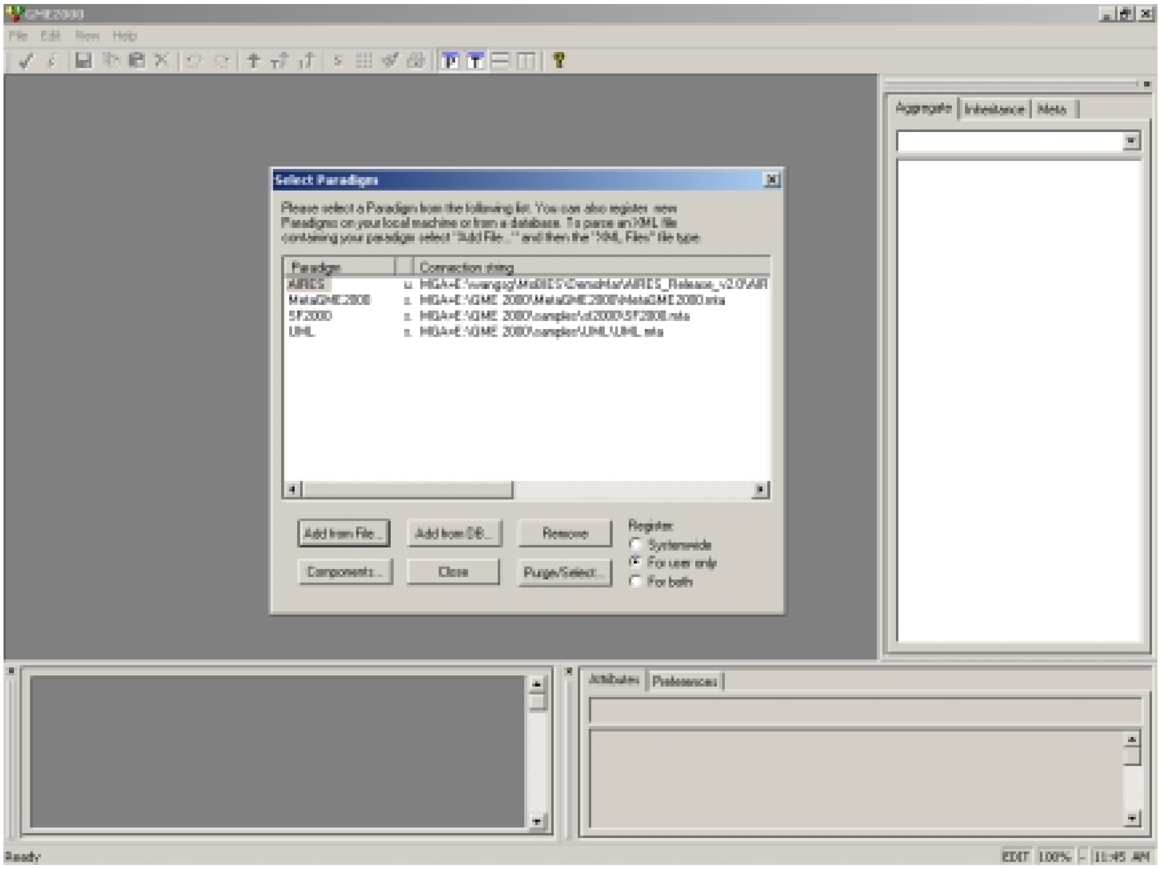
- AIRES meta-model defined in two files (user can use either one of them to registered the meta-model)
  - ❖ A xml file *AIRES_META_MODEL.xml*;
  - ❖ A xmp file *AIRES_META_MODEL.xmp;*
  - ❖ A zipped file *icons.zip* for icons
- AIRES_Release_v2.0 directory contains interpreters:
  - ❖ *Composition.dll*: interpreter for component composition and signal checking;
  - ❖ *Comp2Task.dll*: interpreter for component-to-task mapping;
  - ❖ *Schedule.dll*: interpreter for timing and schedulability analysis, including deadline distribution, timing attribute assignments, task allocation, schedulability analysis, and OIL code generation.
- Example directory contains an ETC model as an example:
  - ❖ Three mdl files from OEP: *etc_manager.mdl*, *etc_monitor.mdl*, and *etc_servo_control.mdl*;
  - ❖ A csv file for port specifications: *etc_ports.csv*

To make AIRES toolkit fully function correctly, all components except the example should be installed.

2.3 Install AIRES meta-model

The package comes with a meta-model that has to be install in GME for all interpreters and the example work. To install the AIRES meta-model, follow steps below to import the AIRES meta-model:

- Start GME v2.0
- Create a new project by choosing ***File***->***Register Paradigms…***
- In the pop-up **Select Paradigm** window, choose ***Add from File…***
- In the pop-up **Open** window, point to the directory containing *AIRES_META_MODEL.xml*, (for example, in C:\AIRES_Release_v2.0 directory)
- Choose *AIRES_META_MODEL.xmp* shown in the window, then choose ***Open***, a new paradigm should be added into the paradigm window with name **AIRES.** (See Figure 1.)
- Click ***Close*** to close the **Select Paradigms** window after AIRES meta-model is installed.



Please select a Paradigm from the following list. You can also register new Paradigms on your local machine or from a database. To parse an XML file containing your paradigm select "Add File..." and then the "XML Files" file type.

Paradigm		Connection string
AIRES	u	MGA=E:\wangsg\MoBIES\DemoMat\AIRES_Release_v2.0\AIR
MetaGME2000	s	MGA=E:\GME 2000\MetaGME2000\MetaGME2000.mta
SF2000	s	MGA=E:\GME 2000\samples\sf2000\SF2000.mta
UML	s	MGA=E:\GME 2000\samples\UML\UML.mta

Register:
○ Systemwide
● For user only
○ For both

Figure 1. Importing AIRES meta-model

The AIRES meta-model can also be registered using XML file. Refer to *GME v2.0 User's Manual* on how it can be done.

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

2.4 Install Icons for AIRES

AIRES toolkit uses its own icons to represent some components like CPU, input and output ports, etc. To make the icons display in models, the GME needs to know the location of these icon files. To install the AIRES icons, do the following:

- Unzip *icons.zip* file to a directory, for example C:\AIRES_Release_v2.0\icons
- In GME, choose ***File***->***Settings***
- In GME properties window, click ***Add*** for **User Icon Path** or **System Icon Path**
- Choose the correct location of icons in **Select Folder** window, then click ***Open***
- Click ***OK*** after return from the location selection window.

2.5 Install AIRES interpreters

AIRES toolkit has 3 interpreter components. These interpreters have been packed as a dynamic link library (.dll file for Windows), and need to be installed in the Windows Registry to be invoked and function correctly at runtime. To install AIRES interpreters, execute the following steps:

- Create a new project in GME, for example, *ETC_example*, by
    - choosing ***File ->New Project….***
    - Highlight **AIRES** paradigm in **Select Paradigm** window, and click ***Create New…***
    - In **New** window, turn on the **Create project file** option, then click ***Next*>**
    - In **Open** window, type project name *ETC_example* in **File name** space, choose **files of type** as ***MGA Files***, then click ***Open***.
    - The ETC_example project should appear in Browser window at right hand side of GME, as shown in Figure 2.

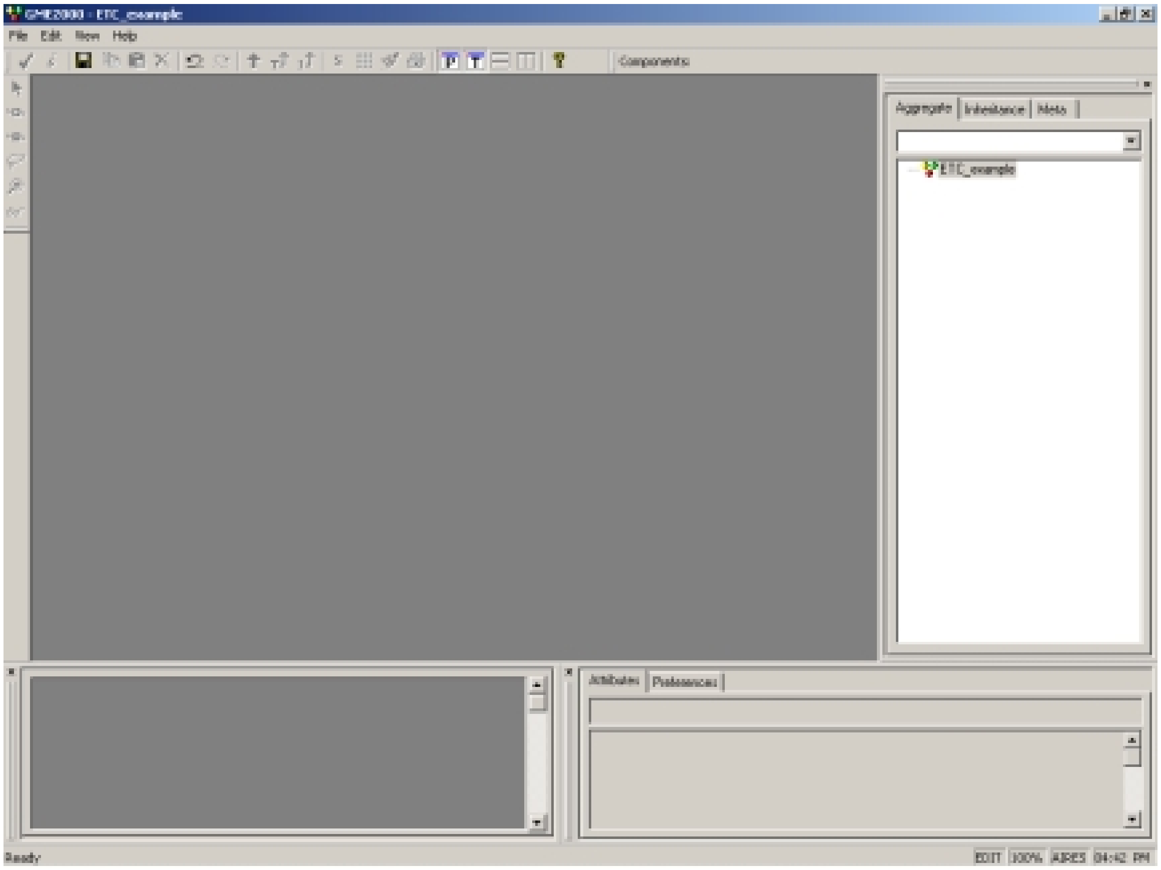
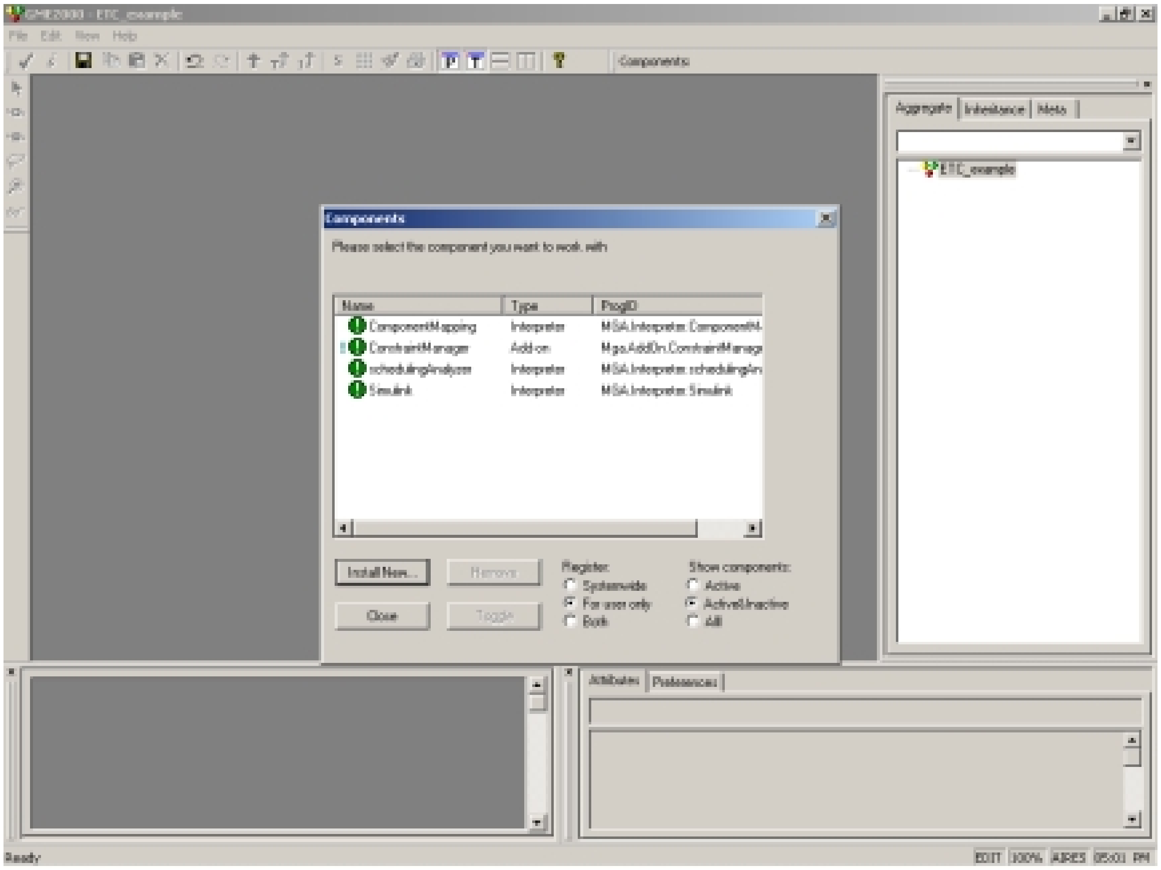


Figure 2. Project ETC_example is created

- Install interpreters
    - Click ***File->Register Components…***
    - In **Components** window, choose ***Install New…***
    - Point to correct location of interpreter components in **Open** window, e.g, *C:\AIRES_Release_v2.0\AIRESComponents*
    - Select **Files of type** to be ***Component Files***, and choose *Composite.dll* in the list.
    - Click **Open** to install composition interpreter. A new line with name **Simulink** will appear in **Components** window.
    - Repeat step 2 – 5 to install interpreters for **Comp2Task** (using file *Comp2Task.dll*), **Schedule** (using file *Schedule.dll*.
    - All interpreters are shown in Figure 3.



Please select the component you want to work with

Name	Type	ProgID
ComponentMapping	Interpreter	MGA.Interpreter.ComponentM
ConstraintManager	Add-on	Mga.AddOn.ConstraintManag
schedulingAnalyzer	Interpreter	MGA.Interpreter.schedulingAn
Simulink	Interpreter	MGA.Interpreter.Simulink

Register:
○ Systemwide
● For user only
○ Both

Show components:
○ Active
● Active&Inactive
○ All

Figure 3. AIRES interpreters are installed

- Choose Close to close the Component windows. Three new icons will appear in the toolbar, as shown in Figure 4.

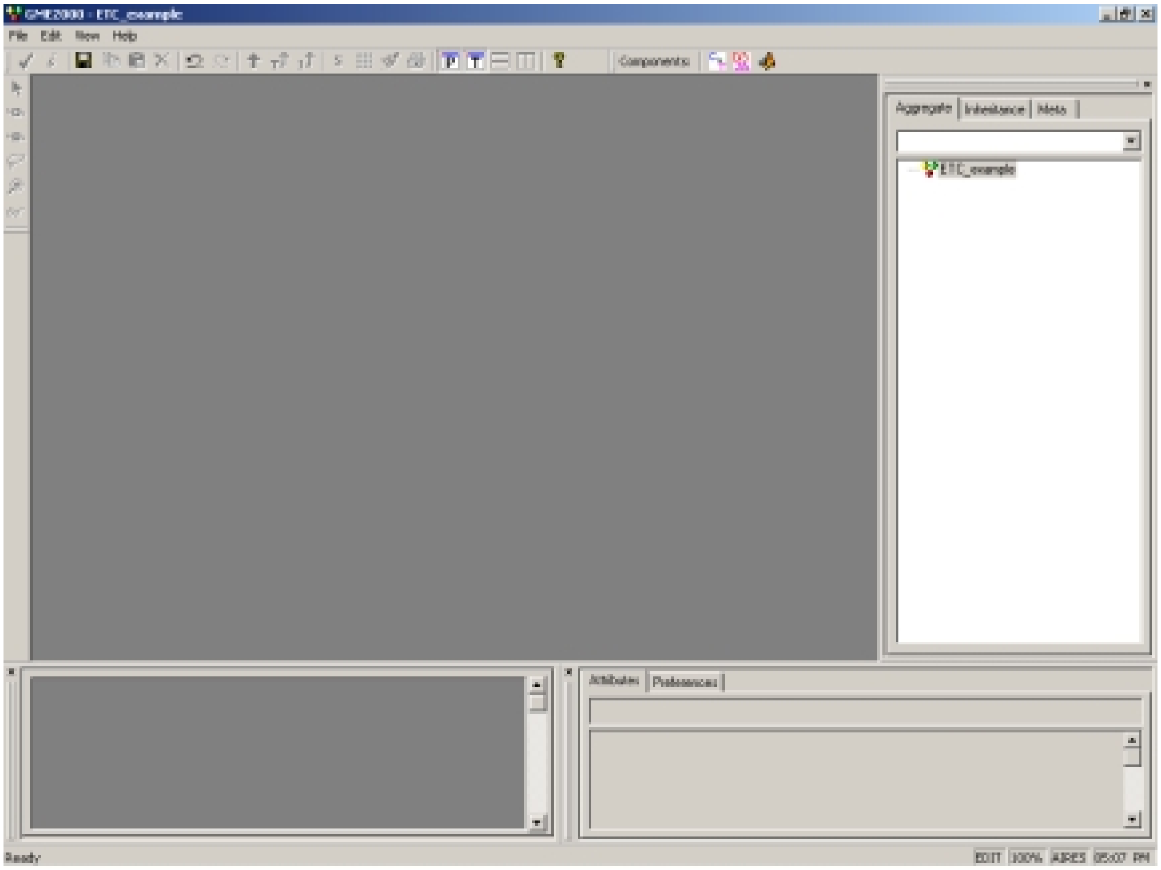


Figure 4. Interpreters shown in toolbar.

The interpreters can then be invoked by clicking corresponding icons. Here is a list of icon - interpreter pair.

Icon: interpreter for importing Simulink models and composition checking;

Icon: interpreter for component to task mapping;

Icon: interpreter for timing and scheduling analysis.

Note that all AIRES interpreters can be installed individual, and work stand-alone. A user can decide which interpreter is desired, install and run it as needed without other interpreters installed.

2.6 Install Graphviz Package (optional)

AIRES toolkit uses a free software package for graphical display of results of component to task mapping. Installation of this package is optional, and won't prevent the interpreter from functioning correctly. The result of component to task mapping interpreter will be displayed as text output in an output window if Graphviz software package is not installed on the machine.

To install Graphviz, downloaded the software package from:
http://www.research.att.com/sw/tools/graphviz/
Then follow the instructions released with the package.

2.7 Rebuild the AIRES interpreters (optional)

The source code of all algorithms/interpreters built in AIRES toolkit are available freely upon request from AIRES group at The University of Michigan for any group in DARPA MoBIES program. Anyone who has the source code can modify them freely for their own purpose. To rebuild the interpreters, the following software should be installed before the rebuild can be done:

- Microsoft Visual Studio tools for Visual C++ 6.0 or above;
- GME 2000 v2.0

To rebuild the interpreters, follow these steps:

- Unzip the source code for different interpreters to different directory.
- For each interpreter, open the project file BONComponent.dsw in MSVC.
- Change the project settings in MSVC Project menu (refer to MSVC user guide). For each interpreter rebuild, one needs to point to correct GME interface location in the ComponentConfig.h file. This can be done by either manually modify the line for this location in ComponentConfig.h file, or run ConfigureComponent.exe tool released with GME 2000 v2.0.
- Choose Rebuild All in Build menu of MSVC.

After the dll file is successfully generated, the interpreter will be automatically registered in the system. So installation procedure described in Section 2.4 need to be performed.

Rebuilding Composite.dll requires a xerces-c_1_2.dll to be located in the same directory where the target dll file will be. This file will used by the window register management tool to correctly perform the interpreter register.

3 Model Construction and Analysis With AIRES ToolKit

3.1 Basic Concept

AIRES toolkit allows designers to import software component models from other modeling tool (e.g., Simulink/Stateflow), reuse them for constructing different applications, 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 model translation and reuse, signal composition checking, and timing and scheduling 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. Components modeled in other modeling tools can be imported into AIRES toolkit to form a reusable repository. A model for an application can then be constructed using the components in the repository.

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 should have the following three folders as shown in Figure 5:

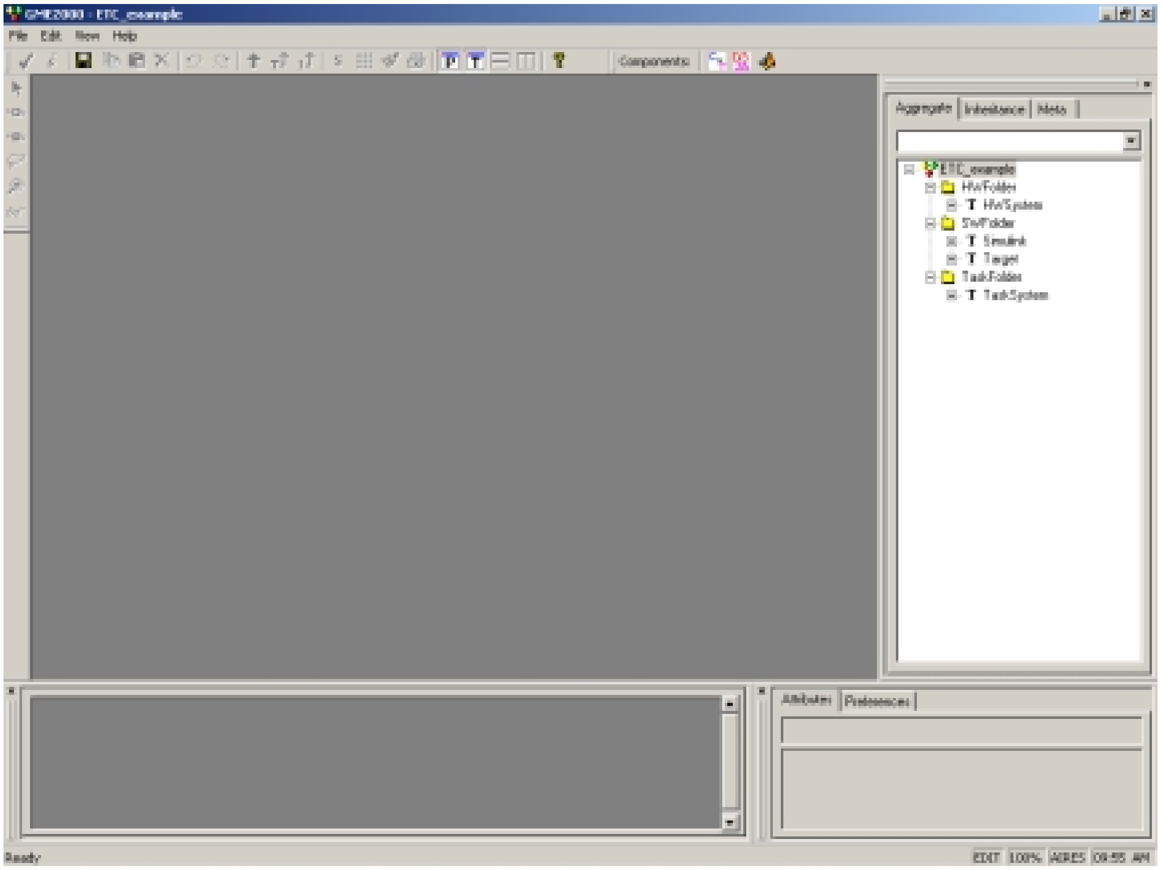


Figure 5. AIRES tool working folders

Hardware component folder: The hardware component folder, **HWFolder**, contains a model of the platform configuration, including the processors, interconnection network, and operating systems on each processor. Figure 6 gives an example of platform configuration with 2 processors connected via CAN bus and both running with OSEKWorks.

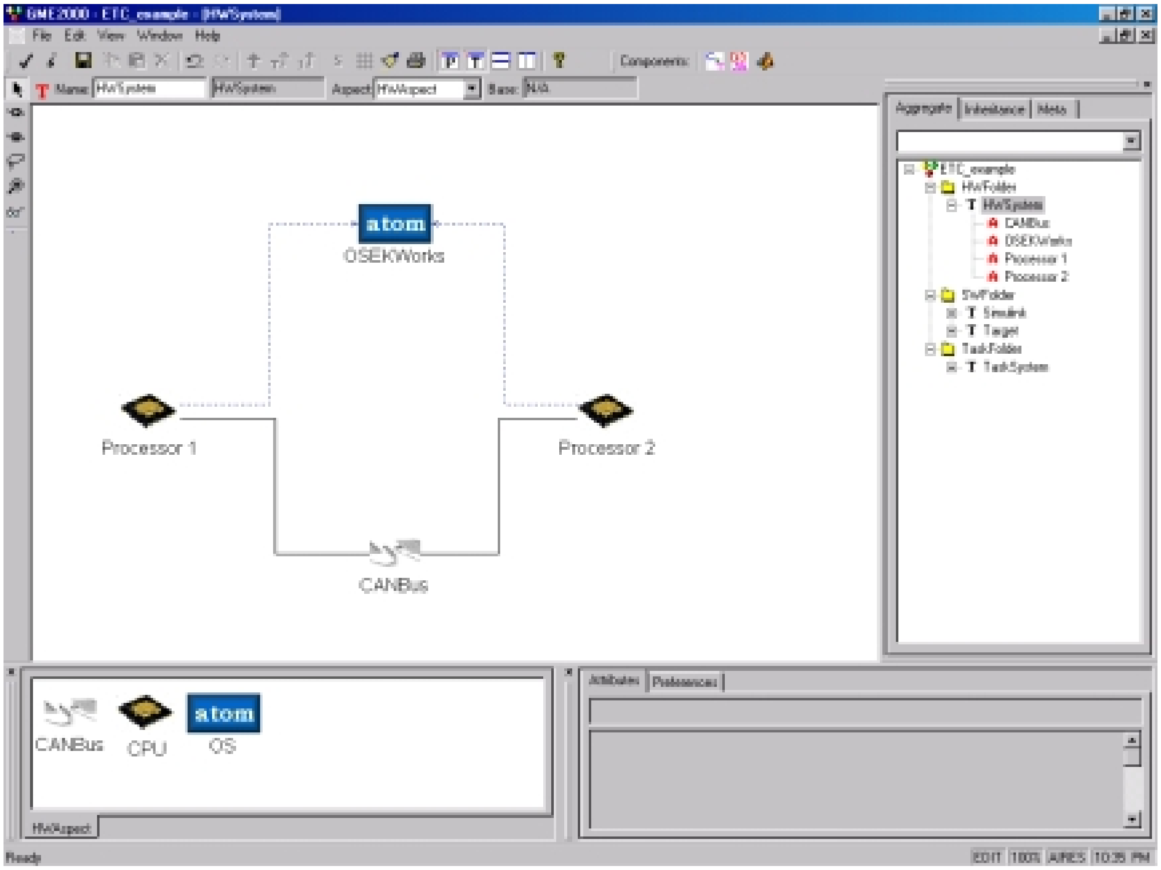


Figure 6. Example of platform configuration model.

Among the hardware components, the OS component has a set of attributes that can be specified by designers and will be used for later analysis. These attributes include timer overhead, context switch overhead and scheduling overhead. The values of these attributes can be obtained through system profiling/benchmarking or other analysis.

Software component folder: The software component folder, **SWFolder**, contains the software components or modules that make up the application. The software components in this folder may either come from some other modeling tool, or be created in GME manually. A model in **SWFold** can be created to store all the available components for a family of applications, therefore forms a component repository for reuse. A model called Target has to be constructed for an application, i.e., the Target model is an application model. The target model consists of interconnected components from both repository and parts window to form a graph. Attributes like component type, communication cost between components can be specified in such a graph. The runtime architecture generation achieved by Comp2Task interpreter will apply only to the Target model. An example of **SWFolder** is shown in Figure 7, which contains 2 models, **Simulink** and **Target**. The **Simulink** model is used as a repository and contains models imported from *Simulink* modeling tool, while **Target** model is an application model constructed using components in **Simulink** repository and shown in the main window.

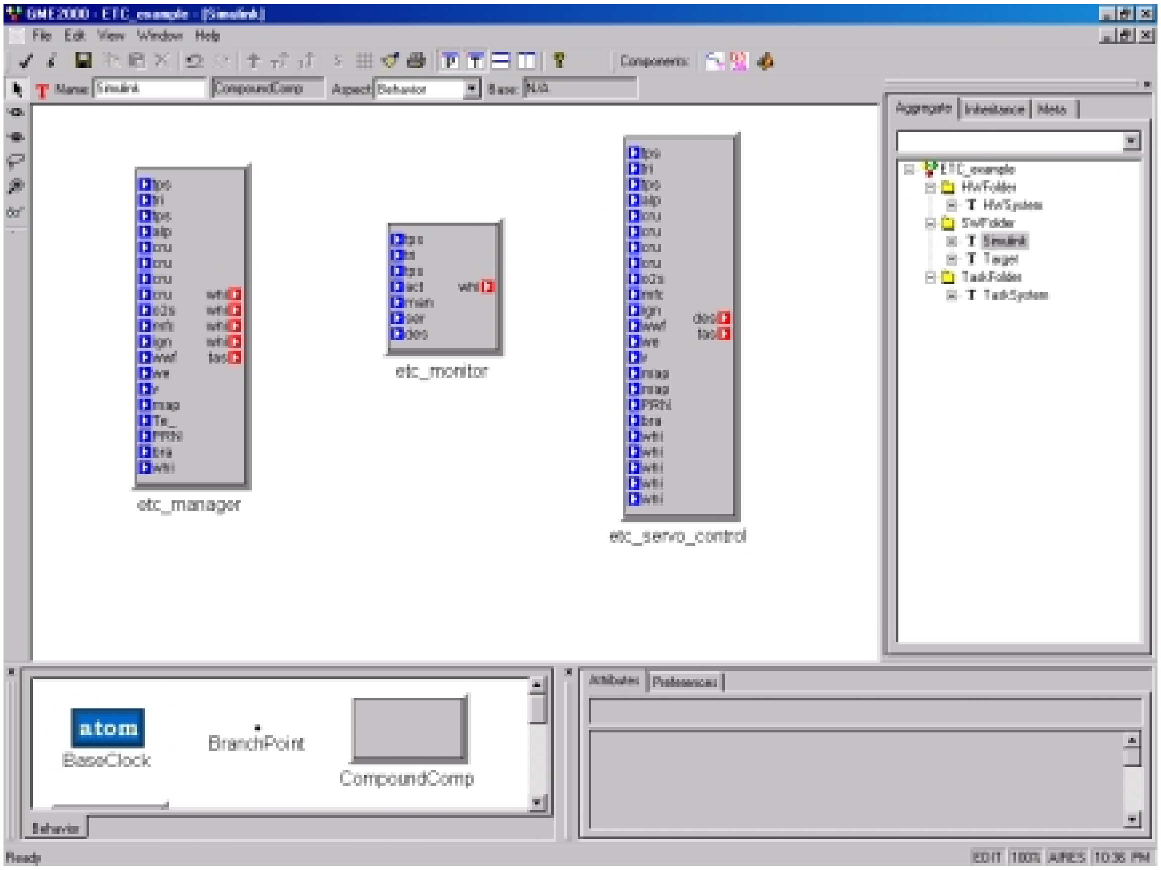


Figure 7. Example of software component folder

Task folder: The task folder, **TaskFolder**, contains a task graph, which is generated according to the results of the component-to-task mapping. Constructing a task model in **TaskFolder** according to the outputs of the component to task mapping interpreter is a manual process in current version of AIRES toolkit. A model in **TaskFolder** contains a view of the application as a set of dependent tasks, which is called runtime architecture. The application model can be further enhanced by specifying the end-to-end timing constraints between tasks and rate of each task. A deadline constraint should also be specified on each output task using the **Constraint** atom in part window to ensure the deadline distribution and timing assignment algorithm working correctly.

The execution time of the tasks and the inter-task communication cost can be derived from the output of the component to task mapping function of the interpreter and is manually input to the model. Figure 8shows an example task graph with timing constraints specified.

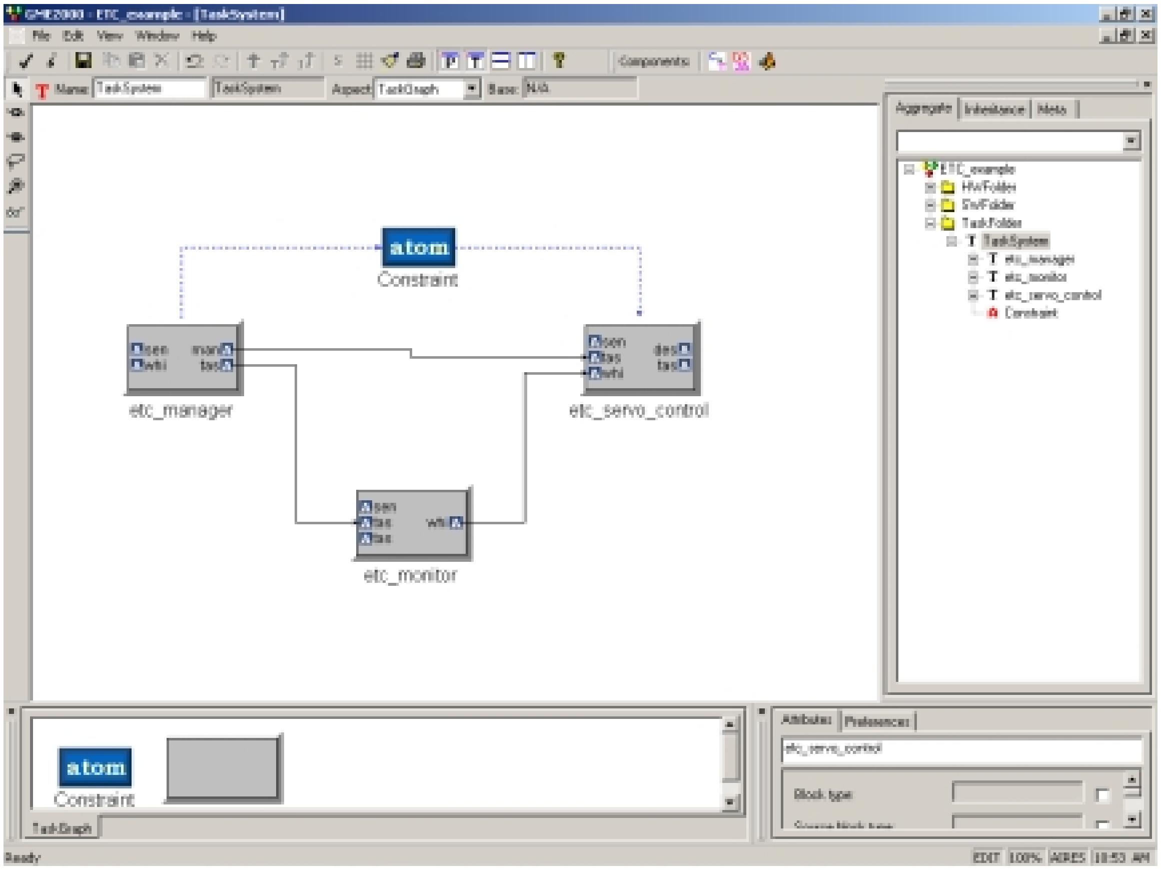


Figure 8. Example of a task graph with timing constraint

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 Component Repository and Design Model Construction

3.3.1 Repository construction

AIRES toolkit supports model reuse. A user can create a set of component models in the SWFolder. These models can be constructed and verified individually, and be used for an application. Such set of models form a component repository for reuse. Figure 9 shows an example of a component repository with models of 3 components in ETC example.

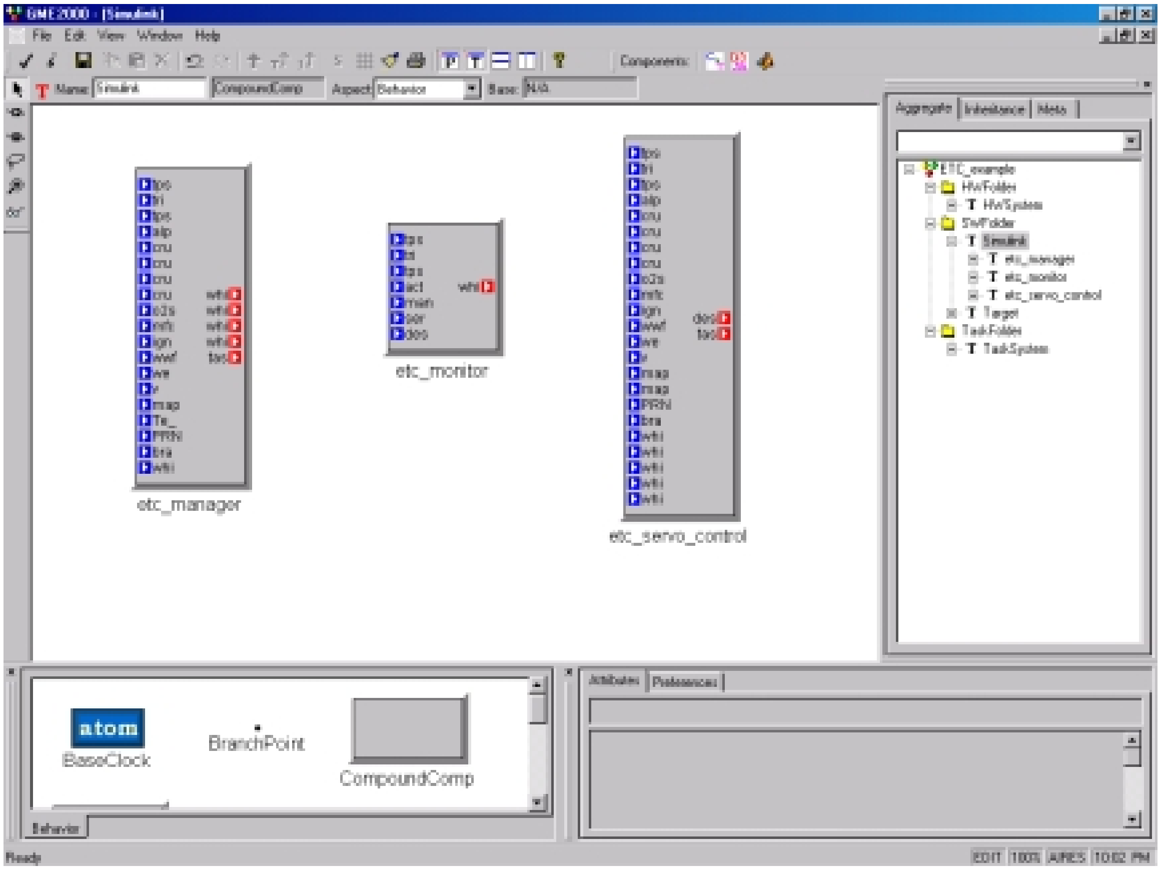


Figure 9. An example of component repository

Users can edit the model of each component in the repository by double clicking the component model in the main window. Since AIRES toolkit focuses on component structure instead of dynamic behaviors, the edition of a component model includes only grouping components to form different granularity and changing port information.

3.3.2 Model translation

AIRES toolkit supports directly importing models from other modeling tools. Now, AIRES model translation supports only directly importing Matlab Simulink block diagrams in .mdl file. The importing is done through invoking Simulink model importing and composition checking interpreter in the toolbar component list. After clicking the Simulink/Composition checking icon, a pop-up window with 3 options: ***Simulink import***, ***Signal import***, and ***Signal check***, as shown in Figure 10.

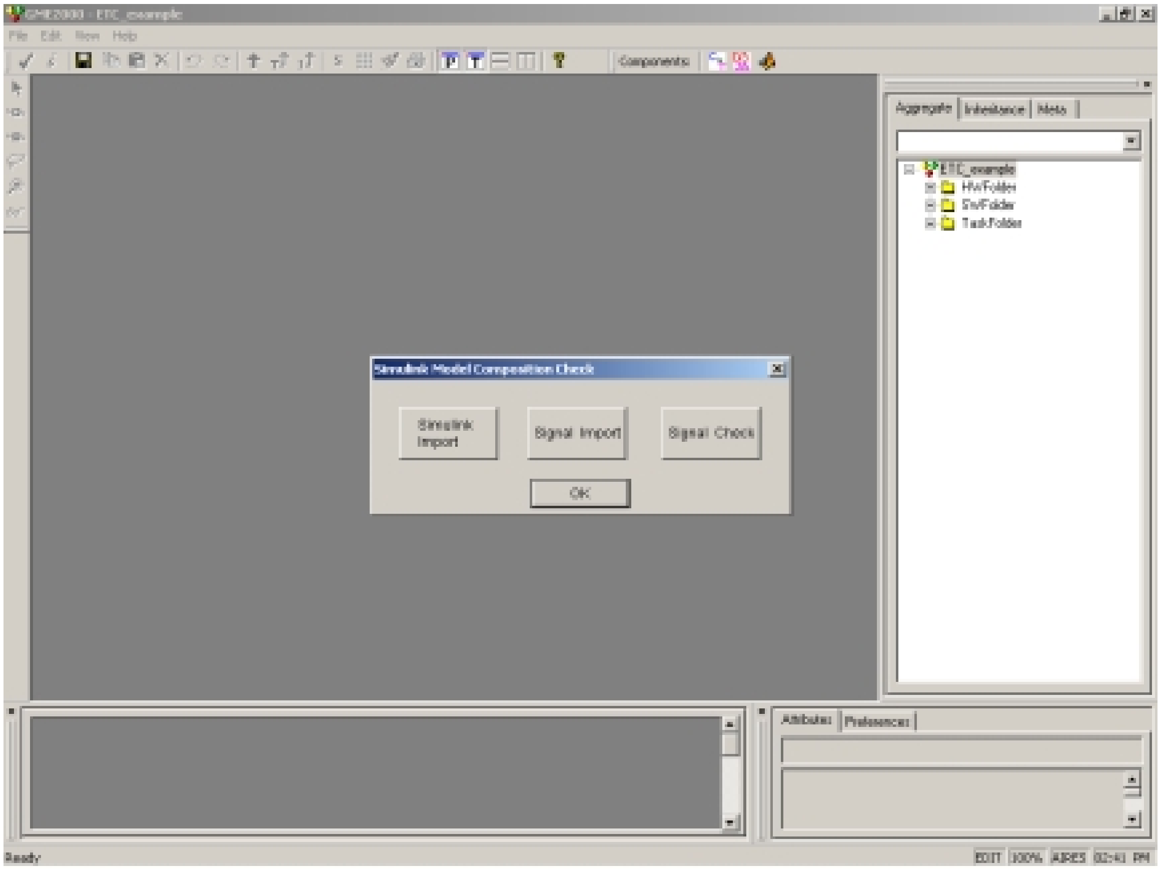


Figure 10. Options for Simulink import and composition check

By clicking ***Simulink Import*** button, AIRES tool will bring up a file option window for choosing the *.mdl* file. After the file is chosen and open, AIRES tool will automatically create a model repository **in SWFolder** with the chosen model listed in it, if the repository does not exist. Otherwise, AIRES tool will simply add the selected model to the current repository. A blank **Target** model will also be generated for application model construction. In our example, the repository is named **Simulink** in **SWFolder**.

3.3.3 Application design model construction

A design model can be constructed by selecting components from the component repository and/or part window, dragging and dropping (or copy and past) them in the blank target model workspace, customizing them with properties (name, input and output ports, data size and frequency, etc.), and linking the ports of different components. The final model should be stored as **Target** model in **SWFolder**. Figure 11 shows an ETC application model constructed by instantiating components in the repository generated from Simulink models.

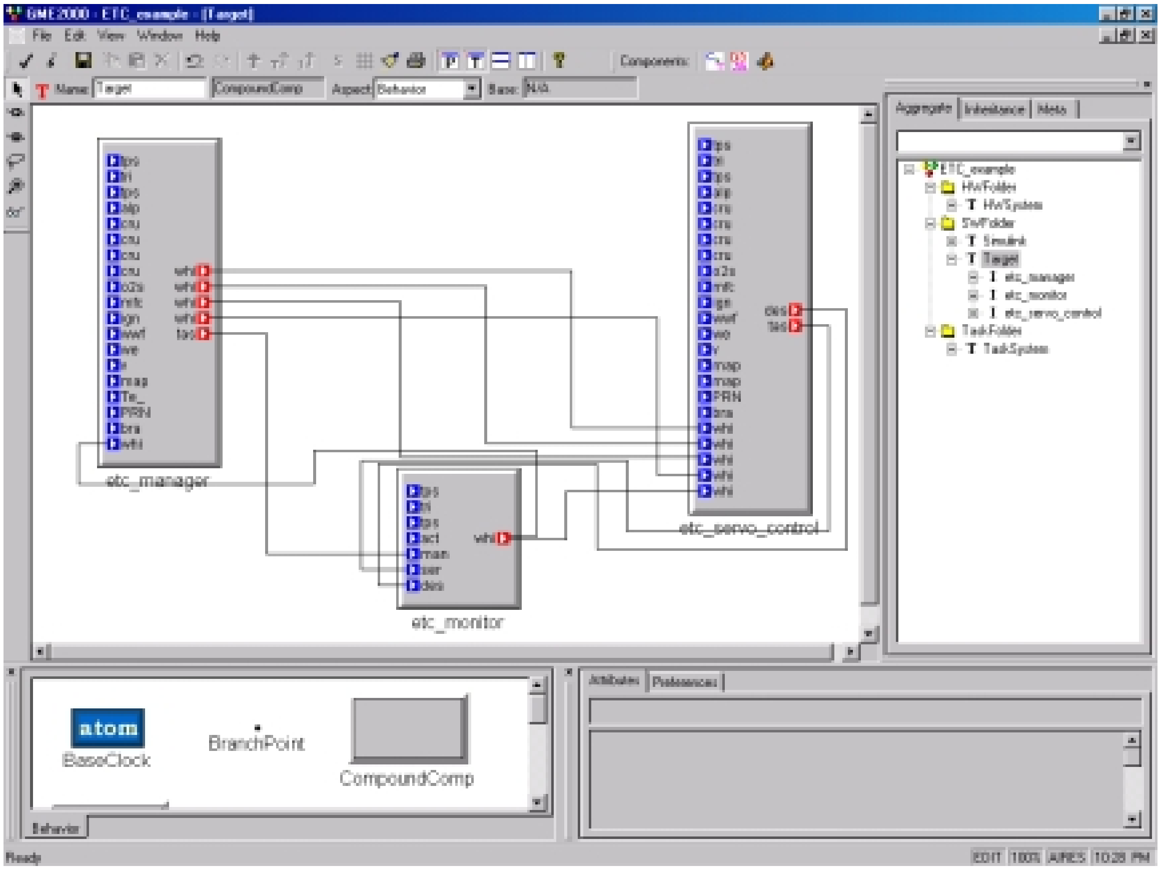


Figure 11. ETC application model

AIRES tool can also check the signal compatibility of the application model. AIRES tool assume signal name uniqueness[1] when performing such checking. To perform the signal checking, port information should be first imported into the system. Such information can be constructed manually in any tool and stored in Excel .csv format. To load the port information into the system, invoking the Simulink and Composition checking interpreter, click ***Signal Import*** button, and choose the right file contains the port information. Then click the ***Signal Check*** button to perform the signal compatible checking. If there exist any inconsistency between signals, a **Signal Composition Result** window will show up to provide the information with those inconsistent signals.

3.4 Runtime Model Generation

AIRES tool support generating runtime model (also called runtime architecture) from design model. The generation is done through the interpreter of component-to-task mapping. The algorithm of component-to-task mapping is based on k-way cuts of a

---

[1] This means if any 2 signals in different components with the same name, these 2 signals should be the same one (come from the same source). For example, tps1 in etc_manager and tps1 in etc_servo_control should link to the same sensor (signal source), while which_fault output from etc_monitor and which_fault input to etc_servo_control should be the same one.

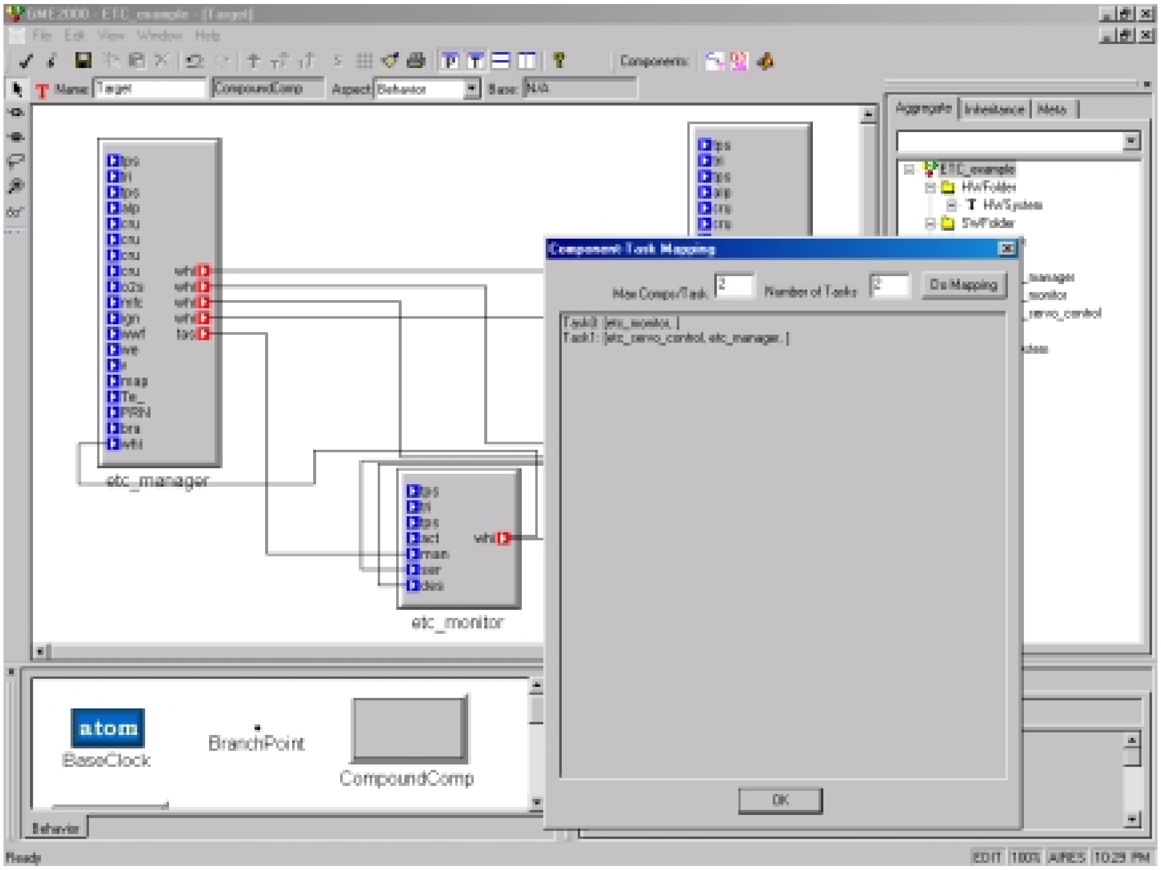
graph. In the current version of the algorithm, minimizing inter-component communication cost is used as a criteria to group components into tasks.

To perform runtime model generation using component-to-task mapping interpreter, an application model should exist in **SWFolder** as **Target**. This model is used as the input of the interpreter. 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 needs to be manually created and stored in Task Folder.

The steps to perform runtime model generation include::

- Invoke the interpreter by clicking the Comp2Task icon in the component list of toolbar. There will be a **Component-Task Mapping** window popped up. Note that the design model has to have already been constructed and stored in the **SWFolder**.
- In the **Component-Task Mapping** window, a designer can adjust two parameters used additionally to the communication cost on how the component will be partitioned into tasks, *Max Comp/Task* and *Number of Tasks.* The parameter *Max Comp/Task* specifies the maximum components can be in a task, while the parameter *Number of Tasks* defines the maximum tasks can possibly exist in the system. The first parameter is designed for balancing the workload among a set of tasks, while the second one is designed for controlling system workload. Currently, *Max Comp/Task* 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*** button to perform component-to-task mapping.

The algorithm outputs both textual and graphic results, if the Graphviz tool is installed. For the ETC example, the result is shown in Figure 12 with given Max Comp/Task = 2 and Number of Tasks = 2.



Max Comps/Task: 2 Number of Tasks: 2

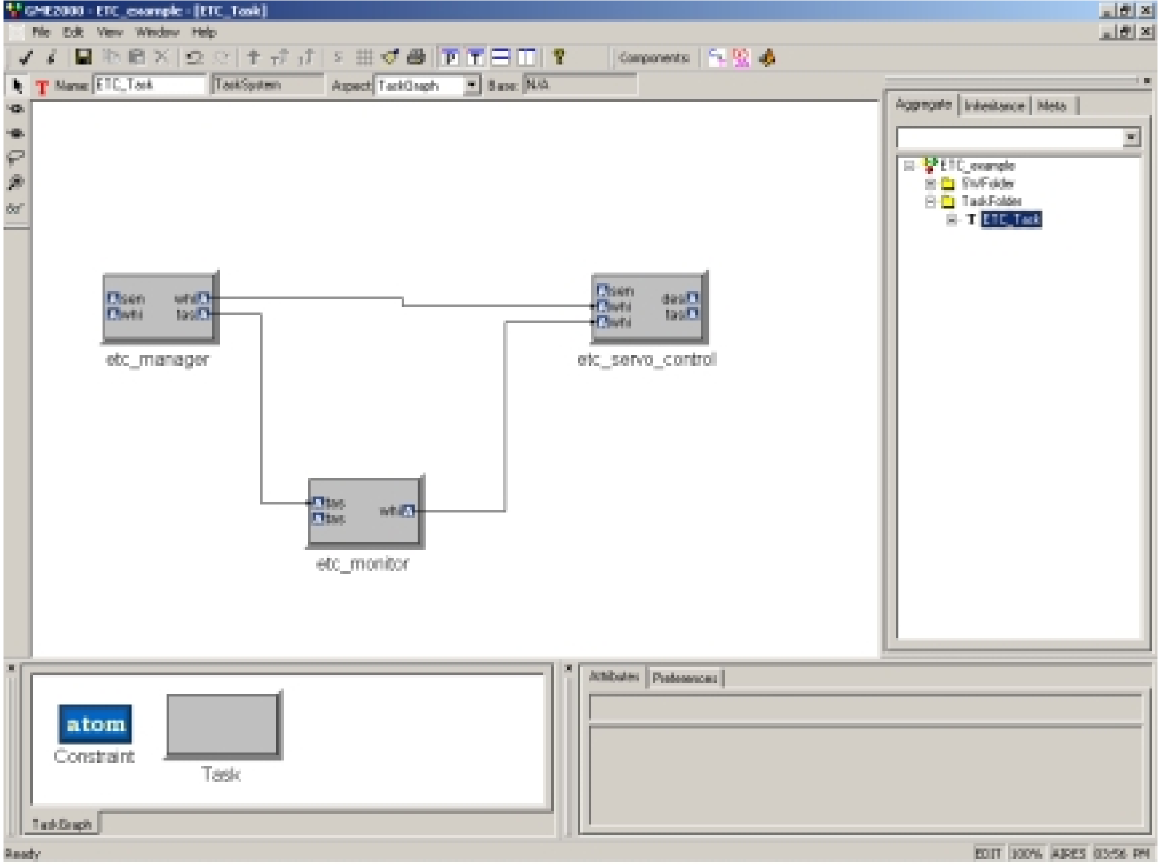
```
Task0: [etc_monitor, ]
Task1: [etc_servo_control, etc_manager, ]
```

Figure 12. An example of component-to-task mapping

The result shown in this example indicates that the software system should be partitioned into 2 tasks: one with only one component, *etc_monitor*, inside it, while the other with 2 component, *etc_manager* and *etc_servo_control*, inside it. If the Graphviz tool is installed, the task graph will be shown in Graphviz, as in Figure 13.

Figure 13. Graphic display of the component-to-task mapping

In current AIRES tool, the runtime model can not be automatically generated according to the output results. A designer has to manually create the task graph accordingly in the **TaskFolder**. A task graph of 3 tasks with each component of ETC in one task is given in Figure 14. Note that this task graph is not created according to the example of our component-to-task mapping results. Instead, it is created according to the ETC specifications given by Automotive OEP ETC example. This also shows that the components of AIRES tool can work independently for different design requirements.



Name: ETC_Task

TaskSystem

Aspect: TaskGraph

Base: N/A

Figure 14. Example task graph with 3 tasks for ETC

3.5 Platform Model Construction

Before any analysis can be performed, a platform model with hardware and operating systems should exist in the hardware folder **HWFolder**. The following steps are required to construct a platform configuration model:

- Create a blank model in the workspace, for example *ETC_Platform*.
- Drag and drop **CPU**, **OS** and **CANBUS** components from the part window in the workspace, and connect them as desired.
- Rename the components in the workspaces to reflect the real part used for the system.
- For OS components, a set of attributes can be assigned, including timer

overhead, scheduling overhead and context switch time. Values of these attributes can be measured individually.

Figure 15 shows an example of platform model with 2 processors both running OSEKWorks operating system and connected through a CAN bus network in **HWFolder.**

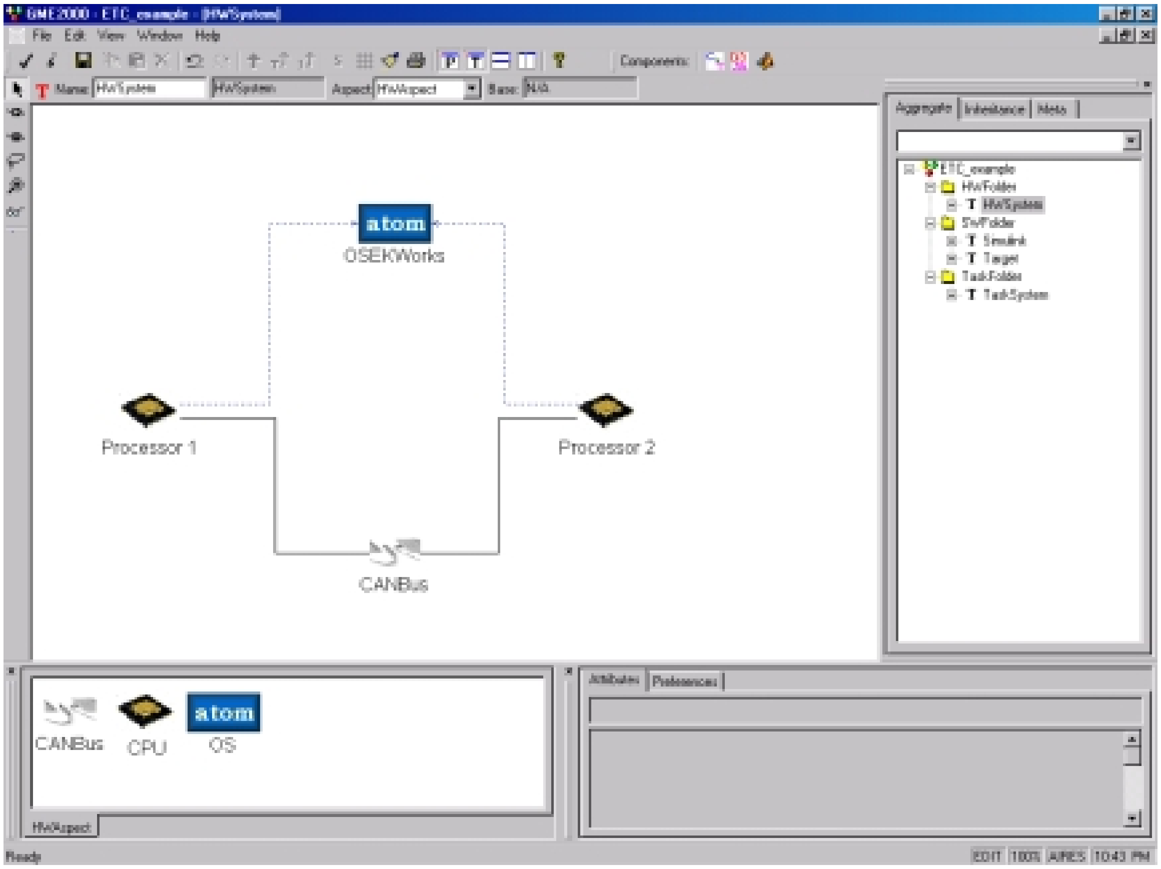


Figure 15. Example platform model for ETC

3.6 Deadline Distribution and Timing Assignments

Deadline distribution is a step to partition end-to-end timing constraints over a set of tasks with/without 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.

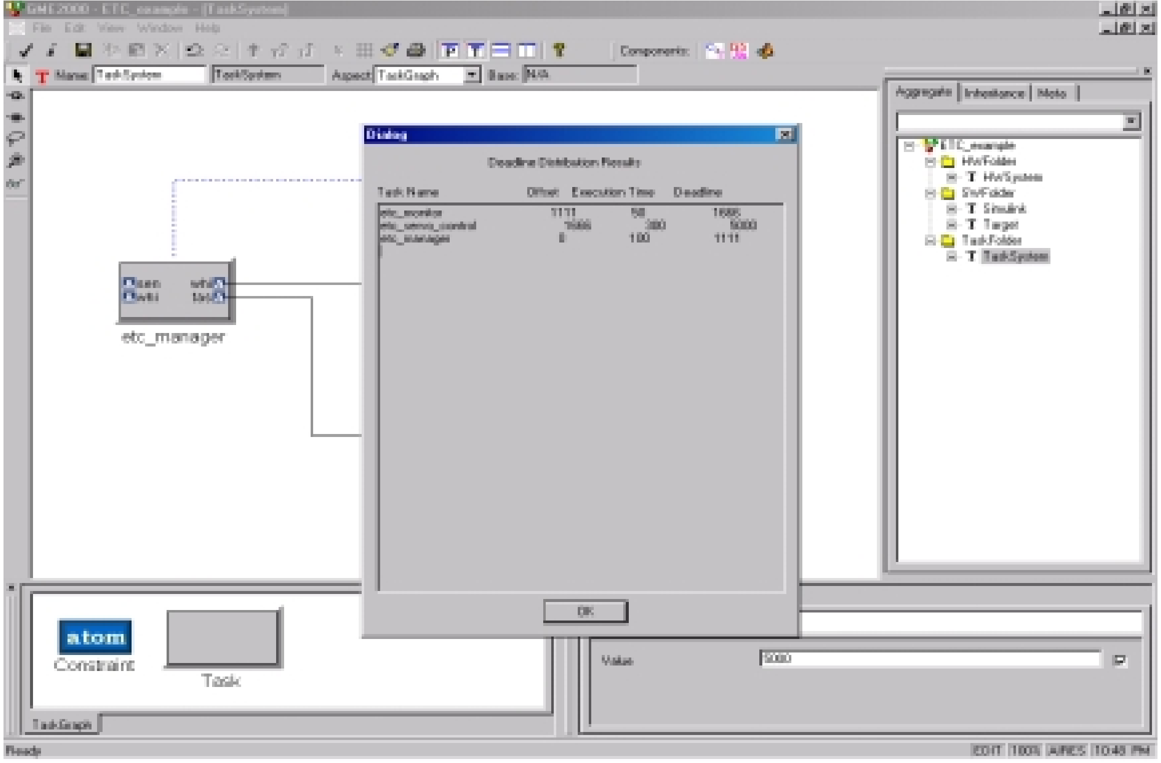
This function of AIRES tool might be an optional step for some embedded system designs in current practices as some timing attributes such as rates and deadlines are already given. For such cases, this step can be omitted.

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

- Specify the timing constraints for the task graph in **TaskFolder** by selecting **Constraint** atom in part window and linking them with tasks at the start and end. Right-click the constraint added to the model to specify the values 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 the deadline distribution and scheduling analysis interpreter icon in the toolbar component list.
- Three options will be shown in the **Real-Time Analysis** window: ***Deadline distribution***, ***Allocation+distribution***, and ***OK***. Click on the ***Deadline Distribution*** button to invoke the deadline distribution.

All tasks in the task graph should subject to some deadline constraints in order to perform the distribution and assignment correctly. If there exists some task that is without any constraint, 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 16 shows the results of deadline distribution and timing assignment for ETC (constraints can be found in the example come with the package).



Deadline Distribution Results

Task Name	Offset	Execution Time	Deadline
etc_monitor	1111	58	1686
etc_servo_control	1686	300	5000
etc_manager	0	100	1111

Name: TaskSystem

Aspect: TaskGraph

Base: N/A

Value: 5000

Figure 16. Deadline distribution results

Note that the later implementation of the system should use to the result given by the deadline distribution in order to satisfy the end-to-end timing constraints as well as the task dependencies.

3.7 Task Allocation and Schedulability Analysis

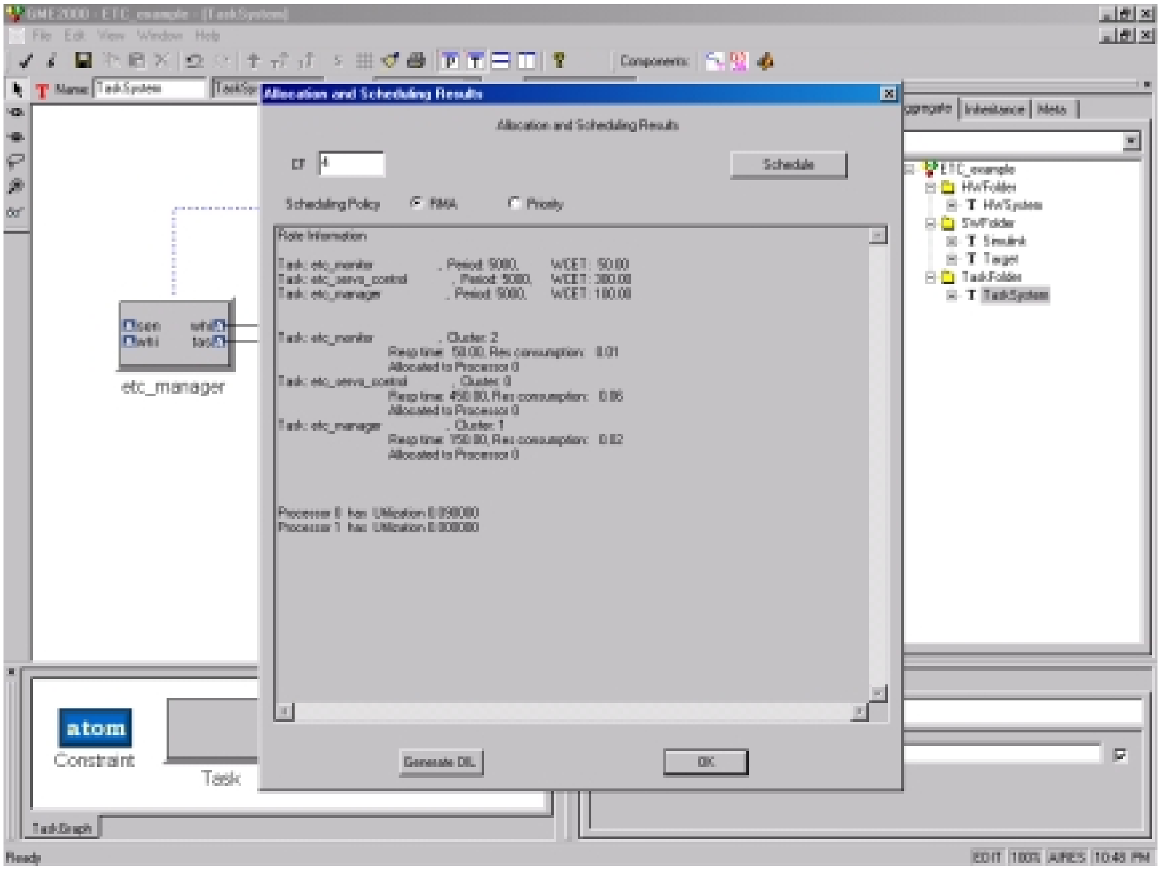
Task allocation is a process to assign tasks to different processors on a platform so that all constraints can be satisfied. Schedulability analysis then checks whether the set of tasks allocated to each process are schedulable. Current version of AIRES toolkit considers both processor utilization and 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 allocate heavily communicating tasks together so that they will run on the same processor to minimize the usage of network bandwidth.

Current version of AIRES tool packs the deadline distribution, task allocation, schedulability analysis and OIL file generation in a single interpreter. To do the task allocation and schedulability analysis, follow the steps below:

- Invoke the schedulability analysis interpreter after the task graph has been constructed in **TaskFolder.** Run ***deadline distribution*** or assign task rate attribute before perform task allocation and schedulability analysis.
- Click on ***Allocation+distribution*** button to run the task allocation and schedulability analysis.
- In the **Allocation and Scheduling Result** window, specify the correlation constant **CF** as needed.
- Choose the scheduling policy used for this analysis. Only RMA is supported in this version
- Click the Schedule button after all selections are done. The analysis results will be shown in the text area of the window.

The results of task allocation and scheduling will show information of individual tasks, including its name, rate and WCET. For each task, the analysis will also indicate which cluster the task is in (a cluster may contain one or more tasks), response time of each task, resource consumption of each task, and the location (processor) on which the task will be executed. The analysis will also provide the utilization of each processor. Figure 17 shows the result for an ETC example with end-to-end deadline=5000 us.



Allocation and Scheduling Results

CF: 4

Scheduling Policy ● RMA ○ Priority

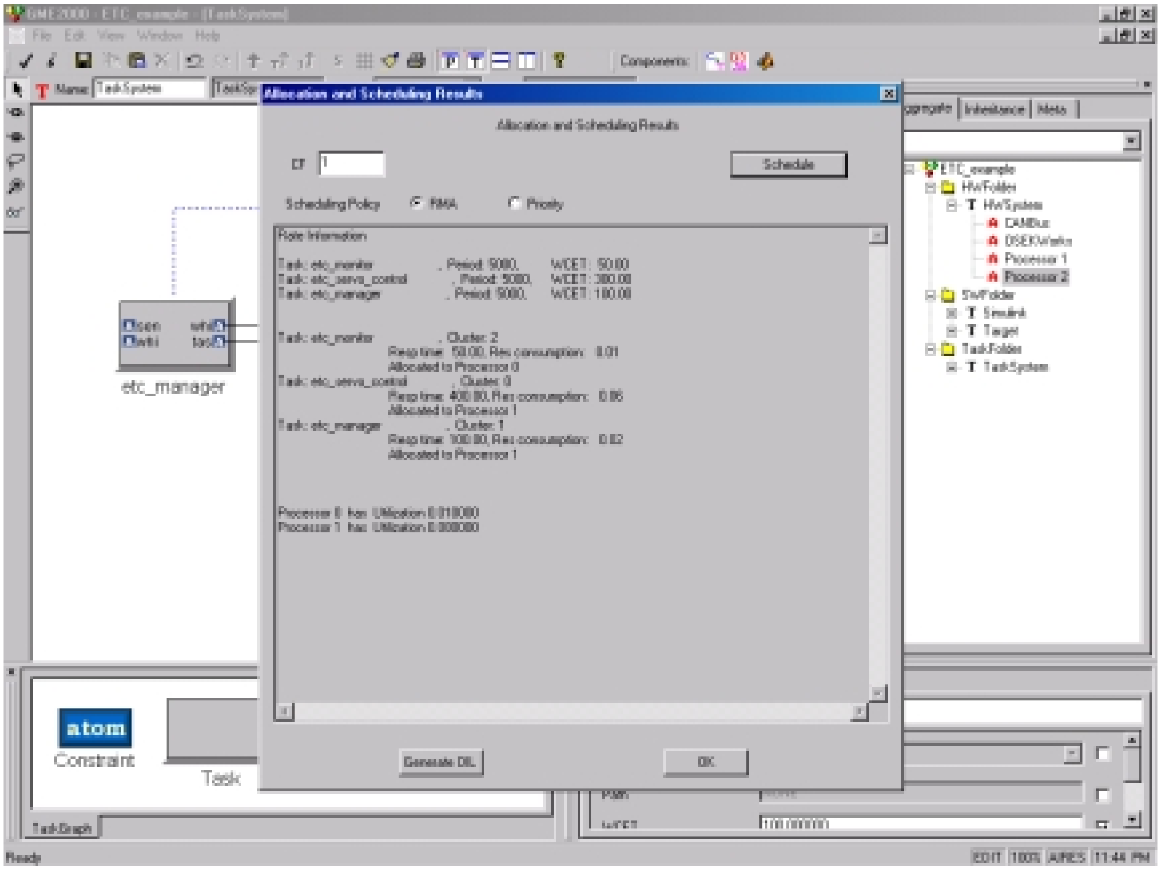
```
Rate Information

Task: etc_monitor          , Period: 5000,     WCET : 50.00
Task: etc_servo_control     , Period: 5000,    WCET : 380.00
Task: etc_manager          , Period: 5000,     WCET : 100.00


Task: etc_monitor               , Cluster: 2
                Resp time:  50.00, Res consumption:   0.01
                Allocated to Processor 0
Task: etc_servo_control          , Cluster: 0
                Resp time:  450.00, Res consumption:   0.06
                Allocated to Processor 0
Task: etc_manager                , Cluster: 1
                Resp time:  150.00, Res consumption:   0.02
                Allocated to Processor 0



Processor 0  has  Utilization 0.098080
Processor 1  has  Utilization 0.008080
```

Figure 17. Analysis results of ETC example

Alternatively, a designer can specify the location of tasks' executions. (This is the case given by the ETC example.) To do so, one can add the CPU reference in each task by

- Enter the allocated task
- Choose a processor in **HWFolder** by right-click it, and select ***Edit*->*Copy***
- Paste the processor in task by right-click and choose ***Paste*->*As Reference***

After all tasks have been allocated, the above steps for schedulability analysis can be performed to generate the scheduling analysis result. Figure 18 shows that the analysis results with tasks are manually allocated on 2 processors (*etc_manager* and *etc_servo_control* are on processor 1, while *etc_monitor* is on processor 2.)



```
Rate Information

Task: etc_monitor          , Period: 5000,    WCET:  50.00
Task: etc_servo_control     , Period: 5000,   WCET: 300.00
Task: etc_manager          , Period: 5000,    WCET: 100.00


Task: etc_monitor               , Cluster: 2
                     Resp time:  50.00, Res consumption:   0.01
                     Allocated to Processor 0
Task: etc_servo_control          , Cluster: 0
                     Resp time: 400.00, Res consumption:   0.06
                     Allocated to Processor 1
Task: etc_manager               , Cluster: 1
                     Resp time: 100.00, Res consumption:   0.02
                     Allocated to Processor 1



Processor 0  has  Utilization 0.010000
Processor 1  has  Utilization 0.080000
```

Figure 18. Analysis results with manual task allocation

3.8 OIL File Generation

Since OSEKWorks operating system is widely used for vehicle control applications, we build an OIL file generator in AIRES toolkit to facilitate the implementation of runtime system. The OIL file is a configuration file in OSEKWorks to specify the system objects, including CPU, task, scheduling policy, alarm, system counter, network and messages, in an implementation. AIRES tool now generates an OIL file with only a subset of objects related to scheduler (CPU, task, alarm and system counter) defined in OIL specifications.

In AIRES toolkit, the OIL file is generated according to the timing and scheduling analysis results. Since the system objects and their attributes defined in an OIL file is closely related to how the tasks are implemented (e.g., explicitly schedule task by calling ChainTask() function call, and explicite task termincation by calling TerminateTask()), we assume that the implementation of each individual task will also follow the information given by the analysis tool, and are built with all mechanisms required for the schedule.

It is now required in current AIRES tool to perform OIL file generation after schedulability analysis, and the generation can only be invoked by clicking the ***Generate***

OIL button in **Allocating and Scheduling Results** window. Following steps are required to generate OIL file:

- Perform schedulability analysis first, as described in Section 3.7.
- Choose ***Generate OIL*** in **Allocating and Scheduling Results** window after schedulability analysis is done.
- In the pop-up file selection window, give a file name for generated OIL file, then choose ***Open***.
- If the OIL file is generated successfully, there is a message window popped up with the full path of the generated OIL file.
- If there are multiple processors in the platform model, the OIL file generator will generate an OIL file for each processor separately.

4 Contact Information

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

- Kang G. Shin (kgshin@eecs.umich.edu)
- Jeong Chan Kim(jchankim@eecs.umich.edu)
- Sharath Kodase (skodase@eecs.umich.edu)
- Shige Wang (wangsg@eecs.umich.edu)