AIRES Toolkit Description

AIRES Group
RTCL/EECS, The University of Michigan

In the package:

The following items can be found in this package AIRES_Release_v2.0.zip:

1. A meta-model: AIRES_META_MODEL.xmp and AIRES_META_MODEL.xml
2. A directory contains all interpreters: AIRESComponents. In this directory, there are 3 .dll files: Composite.dll, Comp2Task.dll and Schedule.dll
3. An ETC example in Example directory. There are 4 files: etc_manager.mdl, etc_monitor.mdl, etc_servo_control.mdl, and etc_ports.csv.
4. A Model_Example directory with a completed example constructed using AIRES: ETC_example.mga is a completely constructed example; two .oil files, Software4P1.oil and Software4P2.oil, are OSEKWorks configuration file generated by AIRES tool with given ETC model.
5. A file contains all icons used by AIRES: icons.zip.
6. AIRES toolkit manual in both MS Word and PDF format: AIRES_Manual_v2.doc and AIRES_Manual_v2.pdf.

Please check all files before starting the example. If there is anything missing in the package, please contact Shige Wang, wangsg@eecs.umich.edu.

Tool components:

AIRES toolkit consists of one meta-model as both xml file and xmp file (used by GME) and 3 interpreters as dynamic link library files. The meta-model defines the modeling framework in which AIRES toolkit will work, especially for timing and resource specification and schedulability analysis. Attributes of each entity used in AIRES toolkit can be found in the AIRES meta-model. The details of each AIRES tool component are listed below:

1. AIRES meta-model
    a. Description: specifies all entities used for modeling and analysis. It needs to be installed in GME 2000 environment.
    b. Input: AIRES_META_MODEL.xmp, or AIRES_META_MODEL.xml
    c. Output: A paradigm in GME, named AIRES
    d. Strength: supports timing and platform modeling
    e. Limitations: only works with AIRES algorithms and in GME environment
2. Component composition
    a. Description: imports Simulink models into GME environment, supports new model construction using these imported models, and performs signal consistency checking. These functions are implemented in Composite.dll interpreter.
    b. Inputs:
        i. mdl files from Simulink for importing models

ii. csv file for port information when signal checking is performed
iii. AIRES paradigm is presented in GME paradigm.

c. Outputs:
i. Components in Simulink model are stored in software directory SWFolder after imported after signal checking.
ii. Inconsistent signals, their names, owner components, and values.

d. Strength:
i. Reuses components constructed in other modeling environments
ii. Signal properties can be specified/modified using a general-purpose tool
iii. Supports semantic checking of signals

e. Limitation:
i. Only Simulink models in .mdl can be imported.
ii. Signals have to be manually specified, instead of automatically extracting them from models.

3. Component-to-task mapping

a. Description: partitions components in a design model into different groups that can be implemented as individual tasks. Such partitioning is based on the communication cost among components. This is implemented in Comp2Task.dll interpreter.

b. Inputs:
i. Target model in SWFolder
ii. Communication cost for each link
iii. Max number of components per task
iv. Max number of tasks in the system

c. Outputs:
i. Tasks
ii. Components in each task
iii. Task graph, if the Graphviz tool is installed

d. Strength:
i. Automatically generates a set of tasks from the design model
ii. Indicates which components should be part of which task
iii. Considers both task workload and system workload

e. Limitation:
i. Only communication cost is considered when constructing a task graph.
ii. A task graph has to be manually generated based on our results.

4. Timing and scheduling analysis

a. Description:

Deadline distribution: distribute end-to-end timing constraints over a given task graph (model), and assign the timing attributes to each task in the graph, if the timing constraints for each individual task are not given. The distribution is based on an equal distribution of slack:

$$s = \frac{D - \sum_{i=1}^{n} e_i - \sum_{k=1}^{m} c_k}{n}$$

where D is the end-to-end timing constraint, e_i is the WCET of task i on the longest path subject to the constraint; c_k is the communication cost between a pair of tasks along the longest path subject to the constraint; n is the number of tasks along the longest path. The offset and deadline of a task x in the task graph can be calculated:

$$o_x = d_{x-1} + c_{x-1}$$

$$d_x = o_x + e_x + s$$

$$o_1 = 0$$

$$d_1 = e_1 + s$$

Allocate the tasks in the graph to processors in a given platform model if the tasks have not yet been allocated. The allocation is based on pair-wise task clustering with clustering factor, CF. Clustering factor, CF, for a pair of tasks, T1 and T2, is given by:

$$CF = \frac{(e_1 + e_2)}{c_{12}}$$

where e_1, e_2 are execution times of tasks T1 and T2 respectively and, c_{12} the communication cost between T1 and T2. Given a threshold of *cf*, tasks with $CF \leq cf$ will be allocated on the same processor.

-For the current release, CF is not used. This is provided as a future extension. Though this value can be derived from the values of execution times, and communication costs between tasks, we have exposed this factor to allow the user a finer grained control over clustering and hence allocation of tasks to processors.

Perform schedulability analysis for each processor, and provide response time and resource consumption for each task. The schedulability is checked with the following equation:

$$W_i(t) = \sum_{j=1}^{i} e_j \left\lceil \frac{t}{P_j} \right\rceil + b_i$$

where $W_i(t)$ is the time demand function for task T_i, e_j and P_j are execution time and period of T_j, and b_i is the blocking time of T_i during execution. The blocking time b_i can be calculated by

$$b_i = \begin{cases} 0, & \text{if full - preemptive scheduling} \\ \max_j(s_j), & \text{if non - preemptive or mixed - premptive scheduling} \end{cases}$$

where *j* is for those non-preemptive tasks T_j with lower priority than T_i but released before T_i, and s_j is the execution time between rescheduling points in T_j. $W_i(t)$ only needs to be checked at times

$$\tau_i = \{l \cdot d_j \mid j = 1...i; l = 1,..., \left\lfloor P_i / P_j \right\rfloor\}$$

The sufficient and necessary condition for a set of tasks to be schedulable on a processor is given as follows:

i. Given a task set $T=\{T_i \mid i=1,..,n\}$ with periods P_i, priorities p_i and deadlines d_i are given, task *i* is schedulable if and only if there is exist a time point *t* such that

$$\min_{t \in \tau_i} W_i(t) \le \left\lceil \frac{t}{P_i} \right\rceil \cdot d_i$$

ii. The whole task set is schedulable on a processor if and only if all tasks satisfy the above equation.

iii. The whole system is schedulable if and only if task sets on all processors are schedulable.

Generate OSEKWorks configuration file, OIL file, with parameters (including task, scheduling option, system counter and alarm objects) result in the system schedulable.

b. Inputs:

i. Task graph (model) in TaskFolder

ii. End-to-end timing constraints for deadline distribution. A value has to be given for each constraint.

iii. For each task in the model, a value of WCET attribute is required. A value of rate can be assigned as an option. No timing assignment will be performed after deadline distribution if a rate value is given. (Note the value for the rate should actually be the period (1/rate).) Other options are not used in current algorithm implementation.

iv. For each link between tasks, a task graph edge weight can be specified as communication cost, and will be used for task allocation. Other options are not used in current version of AIRES tool.

v. Platform model in HWFolder. For an OS component, timer overhead, context switch time and scheduling overhead can be specified. If no value is given for any of these, zero overhead is assumed.

vi. A CF value should be given for allocation.

vii. Scheduling policy for schedulability analysis (generalized RMA for current version)
viii. OIL file location and name for OIL file generation.

c. Outputs:
i. Deadline distribution results: released offset, execution time and deadline assigned to each task
ii. Task information: name, period, WCET
iii. For each task, cluster id, response time, resource consumption for this task, and processor it is allocated are given
iv. Utilization of each processor
v. If not schedulable, a text of task set is not schedulable is given.
vi. For OIL file generation, files in given directory with given names.

d. Strength:
i. Automate the runtime design process, including timing attribute assignments, task allocation, schedulability analysis and configuration code generation
ii. Support analysis in a distributed environment with multiple processors.
iii. Take underlying system overhead such as OS overhead into account during analyses.
iv. Support timing analysis with implicit information, e.g., rate of a task is not given
v. Generate OIL file

e. Limitation:
i. Only work for acyclic-graph. Communications that will introduce cycle in the graph are not taken into account in current analysis algorithm.
ii. Consider only rate-monotonic scheduling policy
iii. Only the configuration file for OSEKWorks can be generated.
iv. A separate OIL file will be generated for every processor that has some task allocated on it.

Obtaining metrics needed for analysis:

For each run-time platform, we need metrics like WCETs, periods for components and tasks in the model. For scheduling analysis, we also need values for operating system overheads like context switch overhead, timing overheads. The WCETs for components and tasks can be estimated using any of the following methods:

1. Profiling the application.

Obtaining precise WCETs within short time is an extremely hard problem and instead of that, we can usually get away with rough hueristics estimates. One possible method is to instrument the application code. One can measure the timestamps (usually obtained using real time clock on the processor) at the beginning and the end of software

component execution. The difference gives the execution time of the component. The trial can be repeated several times to obtain an average execution time for a component. The WCET for the component can be taken as some safety fudge factor times the average execution time. The decision about the factor is done based on previous experience. This would vary from one company to another. For example, GM engineers use 1.5 or 2.0 for the fudge factor.

The alternative to this method is exhaustively find an input for which the software component takes longest time to execute.

2. Using compiler analysis.

This method only works if the code is straight-line code, or code with loops running fixed times. One can follow the longest execution path in the code. Then, given the clock cycles for each instruction executed, one can obtain the WCETs. For this we need to parse the code, obtain a graph representation and then, derive WCETs. Currently we dont plan to take this work as part of AIRES.

For workstations (unix, linux), we could use tools like gprof,prof to obtain a call graph profile and get rough execution times from this. We could use these values as WCETs. This can be done for nearly all processors for which GNU's gcc exists. But for MPC555, or HC08, we aren't sure how this can be done.

Both the above methods require application code. For the operating system profiling, we don't need the OS code. We just need to measure the end-to-end overheads (context switch overheads, timing jitters) experienced by an application. We currently have a methodology on how to obtain this without the source code of the OS. We intend to release the results of profiling OSEK on MPC555 along with the methodology to profile any other OS.

Other metrics like period of tasks, timing constraints on the task graphs can be obtained from Simulink models.

3. Obtaining data for Component to task mapping

We need an estimation of the amount of communication involved between software components. A rough estimation of this can be derived by multiplying the sizes of the data types exchanged by the interface (both inputs and outputs) and the frequency of exchange. This information can be obtained from the Simulink model or the code. The communication data between tasks required for task-to-processor mapping can also be similarly derived.

Number of tasks to map the components to depends on the designer. More granularity (higher number of tasks) increases the operating system overheads, but can make a system schedulable. Lesser granularity decreases the overheads but can result in unschedulable system. This becomes a factor only when the system is large and consisting of hundreds of components. For the ETC example, we don't really have to do Component-to-task mapping as this is already done for us.

Work-out Example:

A lot of tools are provided in AIRES, which help the designer in deriving timing and task structuring information. Some of them are not really required in ETC example as these information are available apriori. So for ETC, no deadline distribution is required, nor is component to task mapping. The tasks structure for ETC can be derived from the P-specs or C-specs ??? of Ford style guide. The software components making up the ETC tasks can also be derived from the mdl files. Only the scheduling and allocation information is not present and AIRES tools suite can be used to obtain this. The other tools can be used when these information are not available or need to be derived from a high level end-to-end timing specifications. The release v2.0 is intended to demonstrate the full capabilities of AIRES, even though some of them are not really necessary for ETC.

To learn and understand AIRES toolkit and its functionalities, we give an example constructed step-by-step to show how AIRES toolkit can be used.

Example: Build ETC controller

1. Construct a ETC design model with components in Simulink/Stateflow diagram given by OEP;
2. Construct a task graph from the design model using component-task mapping
3. Analyze timing and schedulability, and perform task allocation for the task graph
4. Generate OIL file for the example

Steps: Note that during model construction, you should frequently save the file in case something goes wrong. We experienced GME crash occasionally during model construction.

1. Install AIRES meta-model and interpreters (this can be found in the AIRES manual).
    a. Start GME 2000 v2.0;
    b. Choose ***File***->***Register Paradigms?***;
    c. Choose ***Add from File?*** in the popup window;
    d. In the new popup window, point file directory to the working directory, and make files of type to be **Paradigm Files [*.xmp]** (should be default);
    e. Double click **AIRES_META_MODEL.xmp** in the file list, or click it once and then click ***Open***.
    f. A new paradigm AIRES should appear in the Paradigm list.

g.Choose ***Component?***;
h.In the new window, choose ***Install New?***;
i.Point directory to **AIRESComponents**, and set files of type to be **Component File[*.dll,*.ocx];**
j.Double click **Composite.dll** (Choose **Composite.dll**, then click ***Open***). A new interpreter Simulink should appear in the interpreter list;
k.Repeat step h - j to install Comp2Task.dll and Schedule.dll. After then, the following interpreters should be present in the interpreter list: ComponentMapping, scheduleAnalyzer.
l.Click close to return to previous window;
m.Click close again to return from popup window.

2.Install icons used by AIRES toolkit

a.Unzip icons.zip to an icon directory, e.g., C:\AIRES_Release_v2.0\icons;
b.In **GME 2000**, choose ***File->Settings?***;
c.In **GME properties** window, choose ***Add?*** for **User Icon Path**;
d.Point the directory to the icon directory icons**,** double click it;
e.Click ***open*** to return to the previous window;
f.Check the **User Icon Path**, the full icon path C:\AIRES_Release_v2.0\icons should exist;
g.Click ***OK*** to return.

3.Create a new project for the ETC model

a.Create an example working directory for new project, e.g., C:\AIRES_Release_v2.0\MyETCModel;
b.Choose ***File->New Project?***;
c.Highlight **AIRES** paradigm, then click ***Create New***?;
d.Use default option **Create project file**, then click ***Next***>;
e.Enter to directory MyETCModel, and give a file name, e.g., ETCModel. Files of type should be **MGA File**.
f.Click ***Open***. ETCModel should appear in GME 2000 Browser window.

4.Import Simulink component models

a.Click Matlab icon located in the component icon list in toolbar;
b.Click ***Simulink Import*** button;
c.Point the directory to C:\AIRES_Release-v2.0\Example, and set files of type to be **Simulink [*.MDL];**
d.Double click etc_manager.mdl in the file list. Wait until return to GME 2000. Now open **ETCModel** in the browser window, there should be a **SWFolder** created. Inside it should be a **Simulink** model and a **Target** model. Double click the **Simulink** model, the **etc_manager** component with all ports should show up in the workspace;
e.Repeat step a - d for etc_monitor.mdl and etc_servo_control.mdl. Now there are 3 components in **Simulink** model: **etc_manager**, **etc_monitor**, and **etc_servo_control**. These components now form a component repository for a ETC controller construction.

5.Create ETC design model

a.Double click the **Target** model in **SWFolder**, and an empty workspace will show up;

b.Right-click **etc_manager** components in **Simulink**, choose ***Edit***->***Copy***;
c.Right-click in workspace, choose ***Paste Special***->***As Instance***. An **etc_manager** component will show up in the workspace.
d.Repeat step b and c to put instances of **etc_monitor** and **etc_servo_control** in the workspace.
e.Connect corresponding ports: click GME 2000 connect mode icon at in the toolbar at left-hand side to connect the following ports
- Etc_manager/out:which_model --- etc_servo_contro/in:which_model
- Etc_manager/out:which_driving_cruise --- etc_servo_contro/in:which_driving_cruise
- Etc_manager/out:which_limiting_rev --- etc_servo_control/in:which_limiting_rev
- Etc_manager/out:which_limiting_traction --- etc_servo_control/in:which_limiting_traction
- Etc_manager/out:task_manager_cntr --- etc_monitor/in:manager_task_cntr
- Etc_monitor/out:which_faults --- etc_servo_control/in:which_faults
- Etc_monitor/out:which_faults --- etc_manager/in:which_faults
- Etc_servo_control/out:desired_current --- etc_monitor/in:desired_current
- Etc_servo_control/out:task_servo_cntr --- etc_monitor/in:servo_task_cntr

f.Option: click each line of link, and assign CommDataSize in the attribute window. Default size is 1.

6.Composition signal checking
a.Click Matlab icon;
b.Click ***Signal Import*** button;
c.Point directory to C:\AIRES_Release_v2.0\Example, and set files of type to **Port Info File [*.CSV];**
d.Double click etc_ports.csv file in the file list;
e.Click ***Signal Check*** button. This will bring up a window showing all the mismatch signals.
f.To fix the signal mismatch:
- Click ***OK*** to return to previous window;
- Click ***OK*** to return to GME;
- Open etc_ports.csv file in MS Excel, make the following changes:
  - Line 21, etc_manager, which-faults, Col H (max): 5 -> 4;
  - Line 29, etc_monitor, tps1, Col H (max): 1000 -> 2000;
  - Line 38, etc_servo_control, tps1, Col D (data_type): int, Col E (dimension): 1, Col G (min): 0, Col H (max):2000
  - Line 59, etc_servo_control, which faults, Col D(data_type): int, Col E (dimension): 1, Col G (min):0, Col H(max):4
- Save the modifications;
- Redo step a - e. A message with No error should show up.

7.Partition component graph to construct task graph (runtime model)

a.Click C->T icon in component list of GME toolbar

b.Give a value 2 **to Max Comps/Task**, and a value 2 to **Number of Tasks**;

c.Click ***Do Mapping*** (don't press Enter key instead);

d.The result will show up with: **etc_manager** and **etc_servo_control** in **Task 0**, and **etc_monitor** in **Task 1**. (You can repeat this with assign different values of **CommDataSize** for each link in Target model, and see different mappings.)

e.To match OEP example (3 tasks), change the value of **Number of Tasks** to 3, and click ***Do Mapping***. The result will show 3 tasks with each contain one component.

f.Construct task graph according to this output. This has to be a manual process now, as described below.

g.Click ***OK*** to return to GME;

h.Right-click **ETCModel** in GME Browser window;

i.Choose ***Insert New Folder->TaskFolder***;

j.Right-click **TaskFolder**, and choose ***Insert New Model->TaskSystem***;

k.Highlight **TaskSystem** model by clicking it, and click it again to change the model name to **ETC_Task** (Note it is not a double click in this step. You can leave the model name as **TaskSystem**, or you can modify it through right-click it, then choose ***Properties***);

l.Double click **ETC_Task** model, and an empty workspace will show up.

m.Drag **Task** part in the part window into the workspace, double click it in the workspace, change the name in attributes window to **etc_manager**. Check the checkbox of **WCET**, and give a value of **200**.

n.Drag 2 **InputPort**s into the workspace, and rename them to **sensor_input** and **which_faults**;

o.Drag2 **OutputPort**s into the workspace, and rename them to **which_output** and **manager_cntr**. Note in this case, we combine the input signals for **etc_manager** component and output signals from it.

p.Go back to **ETC_Task** model by clicking Parent icon in toolbar;

q.Repeat step m - p for **etc_monitor** task and **etc_servo_control** task.

- Etc_monitor: WCET - 300; 3 input ports - desired_current, manager_cntr, servo_cntr; 1 output port - which_faults;
- Etc_servo_control: WCET - 800; 3 input ports - sensor_input, which_manager, which_faults; 2 output ports - desired_current, servo_cntr;

r.Link the task to form a task graph:

- Etc_manager/out:which_output --- etc_servo_control/in:which_manager
- Etc_manager/out:manager_cntr --- etc_monitor/in:manager_cntr
- Etc_monitor/out:which_faults --- etc_servo_control/in:which_faults
- A value of weight for each link can be assigned as an option

8.Timing and schedulability analysis

a. Drag and drop a **Constraint** in **ETC_Task** workspace, rename it as **ETC_cycle_constraint**, give its value as **5000**, and link it from **etc_manager** -> **ETC_cycle_constraint** -> **etc_servo_control**.

b. Create a platform model as follow:

- Right-click **ETCModel** in browser window, choose ***Insert New Folder*** -> ***HWFolder***;
- Right-click **HWFolder**, and choose ***Insert New Model*** -> ***HWSystem***;
- Rename the new **HWSystem** to **ETC_Platform**;
- Double click the **ETC_Platform** model to open an empty workspace;
- Drag and drop 2 **CPU** components, one **CANBus** component and one **OS** component in the workspace. Rename the **CPU**s to **Processor 0** and **Processor 1**, and **OS** to **OSEKWorks**. Link **Processor 0** to **CANBus** and **Processor 1** to **CANBus**, and link **Processor 0** to **OSEKWorks**, and **Processor 1** to **OSEKWorks.** (Note to link **CP**U and **OS**, the connection should always start from **CPU** and end at **OS**. Otherwise, there will be a constraint error.)
- Assign overhead values for **OSEKWorks** component: **Timer Overhead**: 100, **Context Switch Overhead**: 54, **Scheduling Overhead**: 21

c. Click DD/SA icon in component list of GME toolbar to invoke analysis interpreter;

d. Click ***Deadline Distribution*** button in the popup window. The results will show up, including task names, offset of each cycle that the task should start, task's execution time, and its deadline.

e. Click ***OK*** to return to the previous window;

f. Click ***Allocation+Distribution*** button;

g. Turn on **Scheduling Policy** to **RMA**, and click ***Schedule***;

h. The result will show up with:

- All three tasks running at period 5000;
- **etc_manager**: cluster 2, response time 300, resource consumption 0.04, allocated on Processor 0;
- **etc_servo_control**: cluster 0, response time 1658, resource consumption: 0.16, allocated on Processor 0;
- **etc_monitor**: cluster 1, response time 729, resource consumption 0.06, allocated on Processor 0.
- Utilization of Processor 0 is 0.3374, and no task is running on Processor 1.
- Note that the response time for **etc_monitor** is greater than the sum of **WCET**s of **etc_manager** and **etc_monitor**. This is because the OS overhead is taken into account. Same for the **etc_servo_control**.

i. Click ***OK*** to return to previous window, and click ***OK*** again to return to GME. We will redo the analysis with the OEP provide information next.

j. According to OEP example, **etc_monitor** should be running on a different processor. Here we assume both processors run OSEKWorks, which is different from what OEP give. To do so, follow the steps below:
   - Open **ETC_Task** model by double clicking it in Browser window;
   - Double click **etc_manager** in the workspace;
   - Open **ETC_Platform** model tree in **HWFolder**, right-click **Processor 0** in the Browser, choose ***Edit*** -> ***Copy***;
   - Right-click the workspace, choose ***Paste Special*** -> ***As Reference***. A reference of **Processor 0** will appear in the workspace.
   - Return to **ETC_task** model by clicking Parent icon in toolbar
   - Repeat above steps for **etc_monitor** and **etc_servo_control**. For **etc_monitor**, copy **Processor 1** in the **HWFolder/ETC_Platform**, and paste as reference. For **etc_servo_control**, copy **Processor 0** and paste as reference.

k. The rate of each task is also given in OEP example. In **ETC_Task** model workspace, click **etc_manager**, turn on the **Rate** checkbox, and assign a value 10000 to it, then press Enter key. Do same to **etc_monitor** with a value 10000, and **etc_servo_control** with a value 5000.

l. Click DD/SA icon to invoke the interpreter. A Warning will be displayed saying that **etc_manager** has different rate than **etc_servo_control**. Click ***OK*** to dismiss it.

m. Click ***Allocation+Distribution*** button. ***Deadline Distribution*** button will still work, but show the result subject to constraint, which won't be used any more in later analysis. The result will be overwritten by the user-specified values in the model.

n. Turn on **RMA** and click ***Schedule***.

o. Different results will show up, with tasks allocated on 2 processors, different response times, and utilizations of both processors.

9. OIL file generation

a. After the scheduling analysis results are displayed, click the button ***Generate OIL*** at the bottom of the analysis result window;

b. Provide a directory to store the OIL file, e.g., MyETCModel;

c. Provide a file name, e.g., Software4P0.oil;

d. Click ***Open***. A message with the named file created will show up.

e. Repeat this for another processor, generate Software4P1.oil.

f. Go to C:\AIRES_Release_v2.0\MyETCModel, check existence of the 2 OIL files.

There is a pre-constructed example with full models defined in the Model_Example directory released with this software. Please contact us if there is any difficulty during the construction.