



# User Manual

This document provides instructions on how to use the XTEAM modeling and analysis environment. This document does not provide general usage instructions for GME. For help with GME, see the GME documentation at:

<http://www.isis.vanderbilt.edu/projects/gme/Documentation.html>

## 1 Getting Started

This section describes how to install and setup XTEAM.

### 1.1 Install the Software Packages

#### Install GME Version 5.11.18

1. Download and run the install file from:  
<http://www.isis.vanderbilt.edu/projects/gme/>
2. You will have to register with Escher.
3. You may use all the default installation options.

#### Download and compile *adevs* Version 1.3.3

1. Download and extract the source from:  
<http://www.ece.arizona.edu/~nutaro/adevs-1.3.3.tar.gz>
2. Follow the instructions in `adevs-1.3.3\docs\install.html` to build `adevs.lib`. You will need to link this file into your simulation.

#### Download and compile Newran Version 02

1. Download and compile the source from:  
<http://www.robertnz.net/ftp/newran02.zip>
2. Follow the instructions to build `newran.lib`. You will need to link this file into your simulation.

### 1.2 Set up the XTEAM Environment

This section provides instructions for setting up the XTEAM environment in GME, creating a new project, running the simulation generator, and viewing the example and metamodel.

### **Register the XTEAM Paradigm.**

1. Open GME.
2. Choose File→Register Paradigms...
3. A dialog box titled Select Paradigm will appear. Click Add from File...
4. A dialog box titled Open will appear. Browse to the location where you saved xteam\paradigms\FastxADL\FastxADL.xmp and select that file. Click Open.
5. The XTEAM FastxADL paradigm is now registered. Repeat this process for any other paradigms you wish to use.

### **Create a New Project.**

1. Choose File→New Project...
2. A dialog box titled Select Paradigm will appear. Select FastxADL from the list and click Create New...
3. A dialog box titled New will appear. Click Next>.
4. A dialog box titled Open will appear. Browse to the location where you want to store your project, give your project a file name, and click Open.
5. You now have an empty project open.

### **Open the Example Model.**

1. Choose File→Import XML...
2. A dialog box titled Open will appear. Browse to the location where you saved xteam\examples\MobileApp\MobileApp.xme and select that file. Click Open.
3. A dialog box titled Import to new project will appear. Click Next.
4. Choose the location where you would like to save the project file. This is a binary version of the model (as opposed to XML) that GME uses when you have the model open. Click Open.
5. You may see a dialog box appear that says “This model was exported using paradigm FastxADL Version ID: {...} Do you want to upgrade to the current version? Current ID: {...}.” Click Yes.
6. The model is now open.

### **Register the Simulation Generator.**

1. When you have a FastxADL model open, choose File→Register Components...
2. A dialog box titled Components will appear. Click Install New...
3. A dialog box titled Open will appear. Browse to the location where you saved xteam\interpreters\FastxADL2adevsBON2Component.dll and select that file. Click Open.

4. The simulation generator is now registered. Repeat this process for any other simulation generators you wish to use.

### **Run the Simulation Generator.**

1. When you are ready to generate a simulation of your architecture, double click the architecture model to open it.
2. Choose File→Run Interpreter→FastxADL Simulation Generator.
3. The simulation generator will run and you will see a number of C++ source files appear in the directory where your model is saved.

### **Open the Metamodel.**

1. Choose File→Import XML...
2. A dialog box titled Open will appear. Browse to the location where you saved xteam\paradigms\FastxADL\FastxADL.xme and select that file. Click Open.
3. A dialog box titled Import to new project will appear. Click Next.
4. Choose the location where you would like to save the project file. This is a binary version of the model (as opposed to XML) that GME uses when you have the model open. Click Open.
5. You should see a dialog box that says “The XML file was successfully imported.” Click OK.
6. The metamodel is now open. You may view any of the paradigm metamodels in this manner.

## 2 Modeling Elements

This section describes the modeling elements used to create XTEAM models.

### 2.1 Structural Elements

Note: For more information about the XTEAM structural modeling elements, see the xADL Structures and Types documentation at:

<http://www.isr.uci.edu/projects/xarchuci/ext-overview.html#types>



Component

**Components** represent the loci of computation. They either contain a sub-architecture (consisting of components and connectors) or they contain a behavioral model (defined in terms of processes). They are connected to other components and connectors through their contained interfaces. Create component types in an Elements folder, then instantiate those types in architecture models.

**May Contain:** Architecture, Interface, Resource, Process

**Attributes:**

- Description – a generic string that describes the component.



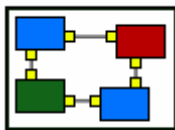
Connector

**Connectors** are the same as components in terms of the way they are modeled. However, connectors are generally used in a different way in an architecture model – components implement business logic, while connectors perform functions such as remote connection establishment, message routing and filtering, etc.

**May Contain:** Architecture, Interface, Resource, Process

**Attributes:**

- Description – a generic string that describes the connector.



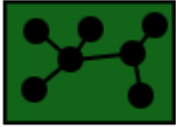
Architecture

**Architectures** represent collections of components and connectors that have been instantiated in a specific topology. Architectures can be used to capture the substructure of components and connectors or the arrangement of top-level components and connectors in a software system.

**May Contain:** Component, Connector, Group, Interface

**Attributes:**

- Description – a generic string that describes the architecture.



Group

**Groups** represent sets of components and connectors. The components and connectors within a group share a thread pool and a FIFO event queue. All components and connectors must belong to exactly one group, and that group must reside in the same architecture as the component or connector.

**May Contain:** Components, Connectors

**Attributes:**

- Description – a generic string that describes the group.
- Thread pool size – the number of threads available to perform tasks for elements in the group.



Interface

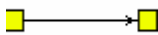
**Interfaces** represent the connection points between components and connectors. Interfaces are connected to other interfaces via links, which represent a logical connection between interfaces, and interface mappings, which denote the realization of a higher-level interface by the interface of a subcomponent. Interfaces are contained by components, connectors, and architectures, and appear as ports on those objects. Interfaces contain inputs and outputs that describe the type of information exchanged over the interface.

Interfaces contain inputs and outputs that describe the type of information exchanged over the interface.

**May Contain:** Input, Output

**Attributes:**

- Description – a generic string that describes the interface.
- Direction – an enumerated type that may be set to none, in, out, or inout. An “in” interface is a provided interface, while an “out” interface is a required interface. An “inout” interface contains both provided and required parts.



A **Link** represents a logical connection between components or connectors over which information is exchanged. The link should be used between components or connectors that exist at the same level within the structural hierarchy. Links are directed, one-way connections – to make a bidirectional connection, create another link in the opposite direction.

**May Contain:** None

**Attributes:** None



An **Interface Mapping** represents the realization of an interface by the interface of a sub-component or sub-connector. The interface mapping should be used between components or connectors that are a different levels of the structural hierarchy (i.e., one is a sub-element of the other). Interface mappings are two-way, bi-directional connections.

**May Contain:** None

**Attributes:** None

## 2.2 Data Elements



Input

An *Input* represents data that is received through an interface. The data type is specified through a contained reference to a datum (called a *PortType*).

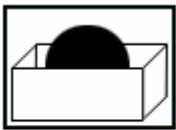
**May Contain:** PortType  
**Attributes:** None



Output

An *Output* represents data that is sent through an interface. The data type is specified through a contained reference to a datum (called a *PortType*).

**May Contain:** PortType  
**Attributes:** None



Resource

A *Resource* represents a data object that is local to a particular component or connector (*i.e.*, only that component or connector has access to it). Resources can be used to store inputs and outputs and maintain the state of a component or connector. The type of the resource data is specified through a contained reference to a datum (called a *ResourceType*).

**May Contain:** ResourceType  
**Attributes:**

- Initialization – a set of C++ statements that initialize the resource.



Datum

A *Datum* represents an object type that is used by components and connectors. Create a reference to a datum within an input, output, or resource to represent a type specification for those elements. Datum objects may be hierarchically contained to create complex types.

**May Contain:** Datum  
**Attributes:**

- Description – a generic string that describes the datum.
- Size – the size of the datum. May be set to an equation involving a random value.
- Type – an enumerated type that specifies the type of the datum. May be set to one of several primitive types (integer, real, string, Boolean) or composite, which should be used when the datum contains other datum objects.

## 2.3 Behavioral Elements

Note: For more information about the XTEAM behavioral modeling elements, please see the Finite State Processes (FSP) documentation at:

<http://www.doc.ic.ac.uk/~jnm/LTSdocumentation/FSP-notation.html>



Process

A **Process** represents a sequence of actions that is carried out by a component or connector. Processes are used to capture the behavior of primitive components (*i.e.*, components that do not have a modeled substructure). XTEAM processes are a form of enhanced finite state processes (FSP).

**May Contain:** Start, Task, Choice, Conditional, Event, ResourceUsage, ProcessTransition, Stop

**Attributes:** None



Start

A **Start** represents the starting point for a process. A thread must be acquired from the thread pool before execution can begin.

**May Contain:** None

**Attributes:** None



Task

A **Task** represents a job that is performed by a component or connector. Tasks are commonly used to manipulate the state of a component or connector or perform some computation. The amount of time required to complete the task is specified by the execution time attribute.

**May Contain:** None

**Attributes:**

- Execution time – the time required for the task to complete.
- Instruction – a set of C++ statements that are performed when the task executes.



Choice

A **Choice** represents a branch in the control flow of a process. It is used to select from among multiple possible control paths based an *external* stimulus (input).

**May Contain:** None

**Attributes:** None



Conditional

A **Conditional** represents a branch in the control flow of a process. It is used to select from among multiple possible control paths based the *internal* state of a component or connector.

**May Contain:** None

**Attributes:** None

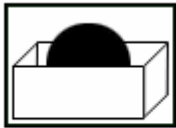


Event

An *Event* represents the occurrence of an input or output. An event is a reference an input or output.

**May Contain:** None

**Attributes:** None



ResourceUsage

A *ResourceUsage* represents the accessing of the data contained in a resource. The access could be either a write (to store an input) or a read (to send an output). A resource usage is a reference to the resource that is being accessed.

**May Contain:** None

**Attributes:** None



ProcessTransition

A *ProcessTransition* represents the branching of control from the current process to a new process.

**May Contain:** None

**Attributes:** None



Stop

A *Stop* represents the termination of a control flow path. A thread that reaches a stop element is returned to the thread pool.

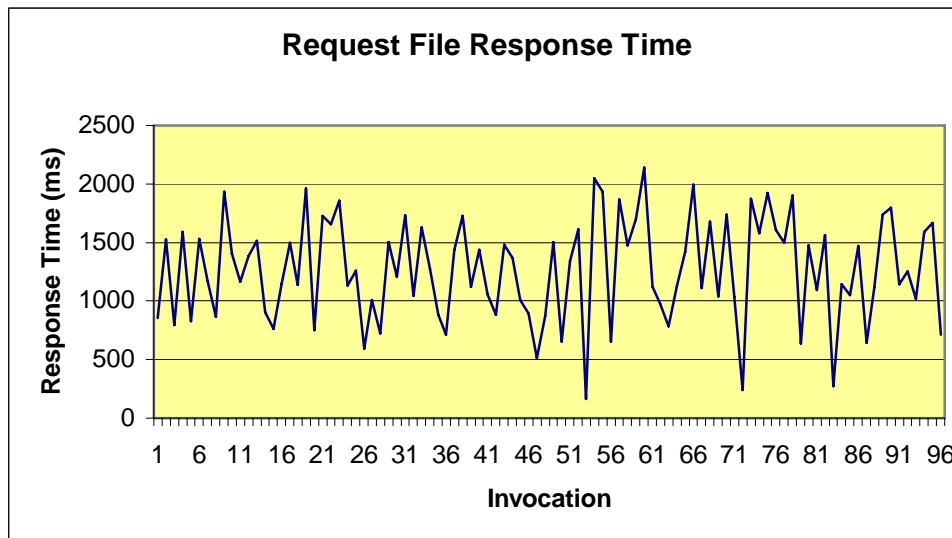
**May Contain:** None

**Attributes:** None

## 3 Performing Analysis

### Using the Latency Extension (FastxADL)

When you generate, compile, and run the latency simulation, a file will be produced for each “out” interface with the name “InterfaceName\_InterfaceID\_Latency\_Log.csv”. You can open this file in MS Excel. Each row in the spreadsheet corresponds to an invocation of the interface. The first column is the time of the request, the second column is the time of the response, and the third column is the round-trip time. Plotting these values yields a graph like the one below:



### Using the Energy Consumption Extension (PowerxADL)

Note: See the paper “An Energy Consumption Framework for Distributed Java-Based Software Systems” by Seo et al. for more information about how the energy consumption estimation works: <http://www-scf.usc.edu/~cseo/publication/usccse2006-604.pdf>.

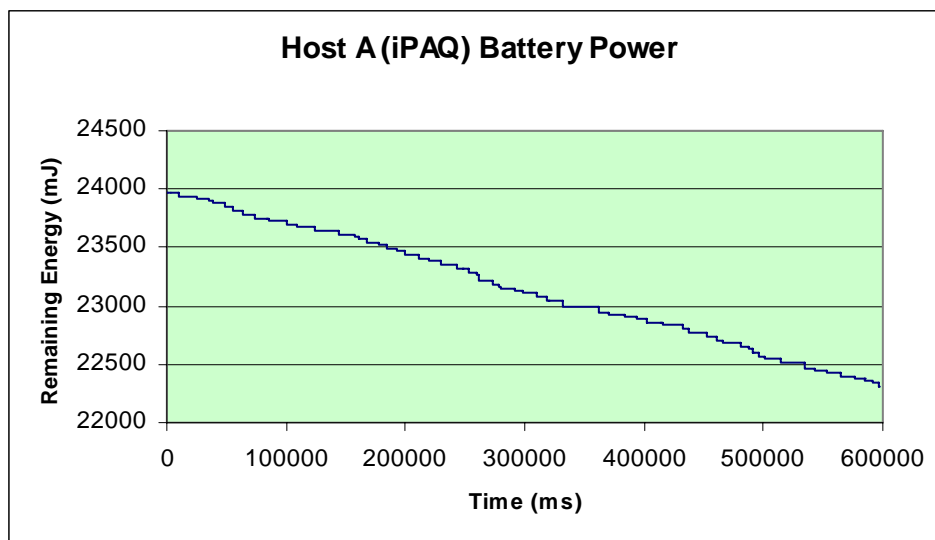
Once you have imported your model into the PowerxADL paradigm, if you have not already done so, create a `Host` element in your top-level architecture corresponding to each host in your system. Choose the set of components that are deployed to each host by clicking on “Set Mode” on the toolbar on the left and then right click on a `Host`. All the elements in the model will become grayed-out. Now, left click on the components and connectors you want to deploy to the `Host`. When you are done choosing the deployment, right click again, and repeat the process for the remaining `Hosts`.

For each `Host`, fill in the following attributes according to measured or estimated values, as given in the paper referenced above:

Transmit energy cost per byte  
Transmit constant energy overhead  
Receive energy cost per byte  
Receive constant energy overhead

For each Interface, fill in the Invocation energy cost attribute with energy cost in Joules of invoking the interface. You may use a constant value, a stochastic value, or some other equation that may include the size and/or value of an input or resource as a parameter.

When you generate, compile, and run the simulation, you will see a new file created for each component and connector named “ElementName\_ElementID\_Energy\_Log.csv”. You can open this file in MS Excel. The first column is the time in milliseconds. The second column is the energy cost incurred by the component at that time. You can determine the total energy used by a component up to a particular time by summing the values in the second column. Subtracting these values from a known battery capacity and plotting them as a function of time yields a graph like the one below:



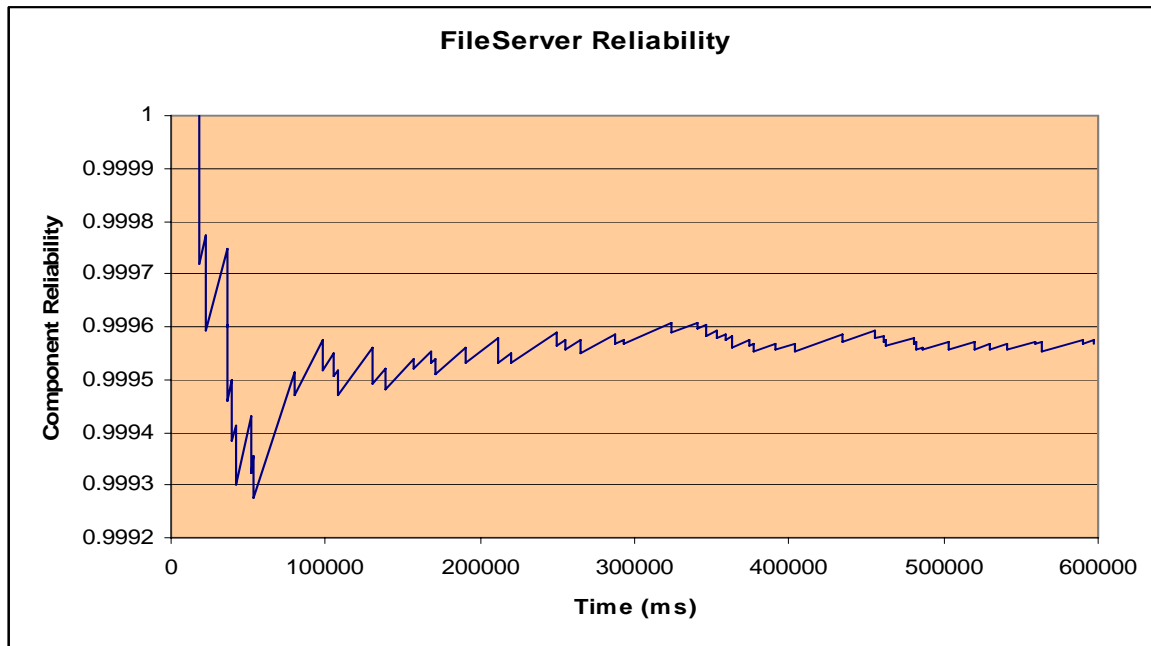
## Using the Reliability Extension (SafexADL)

Note: See the paper “Estimating Software Component Reliability by Leveraging Architectural Models” by Roshandel et al. for more information about how the reliability estimation works: <http://doi.acm.org/10.1145/1134285.1134432>.

Once you have imported your model into the SafexADL paradigm, create a Failure element for each type of failure that can occur during each Task. Fill in the Failure probability attribute with a probability between 0 and 1 that the failure occurs during any single execution of the task. Fill in the Recovery time attribute with the time in milliseconds required to recover from the failure. Create a Fail connection from

the task to the failure, and a `Recover` connection from the failure to the next process executed on recovery.

When you generate, compile, and run the simulation, you will see a new file created for each component and connector named “ElementName\_ElementID\_Reliability\_Log.csv”. You can open this file in MS Excel. The first column is the time in milliseconds. The second column is the fraction of the total elapsed time that the component is not in a failure state. Plotting these values yields a graph like the one below:



## Using the Memory Usage Extension (MicroxADL)

Once you have imported your model into the MicroxADL paradigm, fill in the `Memory usage` attribute for each `Task` with an equation that specifies the amount of memory in KB required to perform the task. You may use a constant value, a stochastic value, or some other equation that may include the size and/or value of an input or resource as a parameter.

When you generate, compile, and run the simulation, you will see a new file created for each component and connector named “ElementName\_ElementID\_Memory\_Log.csv”. You can open this file in MS Excel. The first column is the time in milliseconds. The second column is the amount of memory being used by the component or connector. Plotting these values yields a graph like the one below:

### FileServer Memory Usage

