Software Requirements Specification:
GPU N-Body Integration Toolkit

Prepared by:
Daniel Bagnell
Jason Economou
Rajkumar Jayachandran
Tim McJilton
Gabe Schwartz
Andrew Sherman

Advisor:
Prof. Jeremy Johnson

Stakeholders:
Prof. Steve McMillan
Alfred Whitehead

Contents

1 Introduction

1.1 Purpose

The purpose of this document is to thoroughly describe the software requirements for the GPU N-Body Integration Toolkit (GPUNIT). The document includes an explanation of the purpose and intended use of GPUNIT, as well as a detailed summary of the functional and non-functional requirements of the software.

1.2 Scope

GPUNIT targets users interested in running quantitative astronomical experiments. The software will be usable by physics students and instructors, as well as theoretical and observational physicists. A user of GPUNIT is expected to possess a general understanding of physics.

1.3 Definitions, Acronyms, and Abbreviations

The following terms are used throughout this document and are defined here for convenience.

Term	Definition
AMUSE	Astrophysical Multipurpose Software Environment. A library used to bring together different physics functionality in a way that they can interact with one another.
(Astrophysics) Module	A pre-existing, low-level library designed by an expert in a specific field of astrophysics. Can be used to simulate real-world physics, create a graphical model or parallelize simulations.
Body	An object made up of matter.
Computing Environment	The set of hardware and components available for running simulations.
Diagnostics	The periodic output of quantities of interest that describe the internal physical state of an experiment.
Entity	A representation of a particle, or a grid defining a gas.
Experiment	In the context of our software, an experiment defines the initial conditions of a stellar system, modules involved and connections between modules and diagnostics/logging tools.

Term	Definition
Grid	A discretized representation of some space, can be finite or infinite, uniform or irregular.
GUI	Graphical User Interface. A program interface using visual icons and cues instead of, or in addition to, text-based input. User commands and interactions are performed most often using a mouse, but can also be performed with specified keyboard strokes.
Logging	Output tracing the progress of the experiment through the phases and modules involved in the software.
N-Body Problem	A problem used for predicting the movements of some number (N) of gravitationally interdependent bodies.
Particle	An ideal body that has some quantity of mass but no defined size.
Physical State	Describes the current properties of some entity at a given time.
Property	A value passed into a simulation. Each entity can have 1 to n properties.
Simulation	A piece of software used to integrate and represent a physical system.
Workflow	A sequence of steps in a process. In the context of this document, a workflow refers to a sequence of operations that may fulfill a use case scenario.

1.4   Overview

1.4.1   AMUSE Overview

GPUNIT interacts with the Astrophysical Multipurpose Software Environment (AMUSE) project. The aim of AMUSE is to provide a software framework for large-scale simulations of dense stellar systems. Within this framework, existing code for dynamics, stellar evolution, and hydrodynamics can be easily combined and placed in the appropriate observational context. Without GPUNIT, a user wishing to perform an experiment using AMUSE must create a Python script incorporating various AMUSE modules. This requires the user to posses an understanding of each module and how to use it. AMUSE may also require efficient and well-designed code to be effective, presenting additional difficulties for inexperienced users. References to specific components and features of AMUSE in this document refer to AMUSE version 3.0, the latest release as of the writing of this document.

1.4.2 GPUNIT Overview

GPUNIT is a software solution that eases the creation, execution, and analysis of experiments with AMUSE. The software enables users to design experiments from start to finish using minimal programming. GPUNIT allows the user to decide which modules to use, set initial conditions for the cosmological simulation, run diagnostics on data to do useful calculations and export the simulation's output in several formats. Experiments created using GPUNIT are reusable, allowing future experiments to build upon existing ones.

1.5 Case Studies

The following is three examples of cases where GPUNIT could be used to simplify the creation of a simulation using AMUSE while not holding back others from adding more functionality.

1.5.1 Theoretical Physicist

A theoretical physicist wants to find out why the Archest cluster has many more heavy stars than light stars. He is curious whether it was born this way or evolved to become like this. To find out, he wants to take information from his observations of the cluster and map it to various initial conditions, then see if the resulting simulation agrees with the observations. These initial conditions include the distribution of isophotes within an image of the cluster taken through a telescope. An isophote is a contour on a snapshot of the cluster within which all stars have equal apparent brightness. To set this up, the theoretical physicist opens up GPUNIT and creates a new experiment. He might want to load a separate module he created specifically for this experiment with a Monte Carlo integrator, so he selects it and adds it to the list of possible modules. He connects it to the experiment via the interface and adds other Modules such as the Single Star Evolution (SSE) module. He will be testing this with multiple types of initial conditions, so he selects which variables to pass in as parameters to the experiment by command line. This user also knows exactly what he wants for diagnostics of the experiment, so he adds his created diagnostics tool to GPUNIT's list of diagnostics, selects it, and wires it up to be run. This user will most likely be running his simulation on a super computer so he moves the experiment file he created to the super computer. This user then will add a few runs of his experiment to the queue, with each experiment using different parameters/initial conditions. The user then is able to look at multiple sets of data and run this experiment several more times in an effort to observe how and when this cluster formed.

1.5.2 Observational Physicist

An observational physicist is curious how long it takes for a star cluster to dissolve while orbiting a galaxy. She is interested in whether this cluster collapses or dissolves first. The observational physicist opens up a new experiment with GPUNIT and selects the Stellar Dynamics module Hermite0 and a Stellar Evolution Module called

Single Star Evolution (SSE). She selects the initial conditions of a Plummer model and runs the experiment looking for the boundary where the star cluster collapses versus dissolves by changing values of the initial conditions. The observational physicist then decides to use a different mass distribution model for the initial conditions and, using GPUNIT, removes the Plummer model, replacing it with a Krupa distribution. She then investigates the same concept.

1.5.3 Physics Student

There is a fifth year physics major preparing to write his senior thesis at a university. He wants to explore the origins of mass segregation. Mass segregation describes the process through which higher mass objects move towards the center of the cluster and the lighter objects are on the outside. The student wants to create an experiment and opens GPUNIT. The student then selects a Stellar Dynamics Module called Hermite0, and a Stellar Evolution Module called Single Star Evolution (SSE). He goes on to select initial conditions of a Plummer model and draws connections between the different modules loaded. These connections represent variables passing between modules. The student knows little about N-Body simulation, so he selects a pre-existing density estimation diagnostic tool to observe the positions of the stars in the cluster. Finally, the student runs the simulation and is able to find variables that need to be changed.

2 Functional Requirements

2.1 Requirement Apportioning

Priority 1	This is the highest priority. Requirements designated to be Priority 1 must be functional for the software to be accepted by the customer.
Priority 2	Requirements designated Priority 2 do not have to be completed for the initial release of the software; however, their completion is required for the final release of the software to be accepted by the customer.
Priority 3	This is the lowest priority. Requirements designated to be Priority 3 are not required for the initial release of the software. These requirements are for future releases of the software.

2.2 Module Initialization

The AMUSE code base has a number of existing modules used for the simulation of stellar systems. The existing modules in the AMUSE version 3.0 code base are: BHTree, Hermite0, phiGRAPE, twobody, smallN, octgrave, SSE, BSE, evtwin, mesa, athena, capreole, fi, and gadget2.

2.2.1 The Stellar Dynamics module can be loaded. Priority 1.

2.2.2 The Stellar Evolution module can be loaded. Priority 1.

2.2.3 The Hydrodynamics Interface can be loaded. Priority 1.

2.2.4 Future modules added to the AMUSE code base can be loaded, provided they conform to the template set forth by GPUNIT and implemented by existing modules. Priority 2.

2.2.5 Currently loaded Modules can be unloaded. Priority 2.

2.3 Entity Management

Entity management is the functionality to add/remove entities to/from the simulation as well as query an entity's current state.

2.3.1 An entity can be added before a simulation or after the simulation meets a defined stopping criterion. Priority 1.

2.3.2 An entity can be removed before or after a simulation. Priority 1.

2.3.3 An entity's initial state can be set via properties in the experiment specification. Priority 1.

2.3.4 An entity's state is locked upon beginning a simulation. Priority 1.

2.3.5 If an entity's initial state contains invalid property values, the user cannot begin a simulation. Invalid properties include numerical values out of range and incorrect input for a given data type. Priority 1.

2.3.6 If an entity's initial state contains invalid properties values, the software notifies the user. Priority 1.

2.4 Model Evolution

The entities' states evolve with time throughout the simulation.

2.4.1 **The user can begin a simulation. Priority 1.**

2.4.2 **The user can not pause a simulation. Priority 1.**

2.4.3 **The user can provide criteria for stopping a running simulation. Priority 1.**

2.4.4 **A running simulation stops if a stopping criterion is met. Priority 1.**

2.4.5 **The user can provide a timeout duration as a stopping criterion. Priority 1.**

2.5 Module Interaction

Modules can interact with each other by sending the required data up to the central data model after the module has completed execution.

2.5.1 **Each module or entity can be queried about their current state. Priority 1.**

2.5.2 **Each module or entity can query the state of another module in the experiment via the central model. Priority 1.**

2.5.3 **Each module or entity can be configured to query another module for values needed to simulate the current time step. Priority 1.**

2.6 Diagnostics

Diagnostics about the state of all entities of the experiment are displayed and/or saved to a file.

2.6.1 **GPUNIT has built-in diagnostics such as a body's mass, speed, density, and radius. Priority 1.**

2.6.2 **The user can create custom diagnostics. Priority 1.**

2.6.2.1 **The user can add a custom diagnostic. Priority 1.**

2.6.2.2 **The user can edit a custom diagnostic. Priority 1.**

2.6.2.3 **The user can delete a custom diagnostic. Priority 1.**

2.6.2.4 **The user can rename a custom diagnostic. Priority 1.**

2.6.3 **The user can select the diagnostics about which to be notified. Priority 1.**

2.6.3.1 **The user can select diagnostics from a list of built-in diagnostics. Priority 1.**

2.6.3.2   The user can select a custom diagnostic that they created. Priority 1.

2.6.4   Custom diagnostics must conform to the template required by GPUNIT. Priority 1.

2.6.5   The user specifies when the diagnostics will be displayed. Priority 1.

2.6.5.1   The user can opt to have the diagnostics displayed after a stopping criterion is met for all entities. Priority 1.

2.6.5.2   The user can opt to have the diagnostics displayed after a stopping criterion is met for selected entities. Priority 1.

2.6.5.3   The user can opt to have the diagnostics displayed when the experiment terminates. Priority 1.

2.6.5.4   The user can opt to have the diagnostics displayed for selected entities when the experiment terminates. Priority 1

2.6.6   The diagnostics can be saved to a file. Priority1.

2.7   Logging

The purpose of the logs is to be able to restart an experiment with the output data if there is an error in the experiment or the experiment needs to be stopped and continued at a later time.

2.7.1   Logging can be enabled. Priority 1.

2.7.2   Logging can be disabled. Priority 1.

2.7.3   The user can specify the output location for the logs. Priority 1.

2.7.3.1   Logs can be output to the console. Priority 1.

2.7.3.2   Logs can be output to a file. Priority 1.

2.7.3.3   Logs can be output to a window in the GUI. Priority 1.

2.8   Cluster Management

The user can use the user interface to view activity on nodes in the cluster as well as distribute experiments to them.

2.8.1 Cluster Status View

2.8.1.1 The user can view the CPU usage of all nodes in the cluster. Priority 1

2.8.1.2 The user can view the free memory and total memory of all nodes in the cluster. Priority 1

2.8.1.3 The user can view the running experiments and their status for all nodes in the cluster. Priority 2

2.8.2 Experiment Distribution

2.8.2.1 The user can select a node and run their experiment on it. Priority 1

2.8.2.2 The user can select a node and run a single module in their experiment on it. Priority 2

3 Non-Functional Requirements

3.1 User Input Hardware

The software is operable using a standard keyboard and mouse.

3.2 File Formats

The software uses standard AMUSE output file formats for data, chosen by the user. The available formats are: HDF5, netCDF, Comma Separated Value (CSV).

3.3 Ease Of Use

3.3.1 **The software greatly increases accessibility to the AMUSE code base by allowing non-expert programmers to create astrophysical experiments. The software allows experts to create experiments that can be shared with the scientific community and reproduced on demand.**

3.3.2 **The software's GUI includes standard functionality typical users have come to expect. This includes resizable windows, clickable menu bar, confirm quitting when your document has been modified, minimizable windows, etc.**

3.3.3 **Experiments created and persisted via the GUI can be run by expert users using the standard text-based shell interface.**

3.3.4 **Users are able to choose between running the GUI and using the standard text-based shell interface.**

3.4 Maintanibility

3.4.1 **The software can be extended to support new functionality such as custom modules, user-specified diagnostics, logging and outputs.**

3.4.2 **The software is documented for readability.**

3.4.3 **The public interfaces exposed by the software's underlying code are documented for future development.**

3.4.4 **All attributes and methods of the non-public implementation within the software's underlying code are commented.**

3.4.5 **The formatting of the software's underlying code conforms to industry-standard best practices.**

3.4.6 **The naming conventions present in the software's underlying code conforms to industry-standard best practices.**

3.5 System Evolution

Future versions will support access to more experiment variables and greater parallelization of modules. The system can be expanded in the future to automatically control distribution of parallel resources and computations. As AMUSE grows, the system should be able to support new types of modules and computations so that it is still a useful framework. Visualization tools can be added to the framework so that users can see a graphical representation of their output data.

3.6 Dependencies

GPUNIT is dependent on AMUSE. The AMUSE environment and its own dependencies must be successfully installed prior to use of GPUNIT.
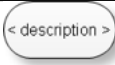
3.7 Security

GPUnit does not create any new security flaws not present in current AMUSE code. Additional flaws are the user's responsibility. If any sensitive data is being processed with this software it is the responsibility of the user to ensure that their communication network is secure enough to protect said data.

3.8 Hardware Requirements

Basic hardware requirements include: a 2.6 GHz processor, 1 GB RAM (2 GB recommended), 256 MB diskspace (512 MB recommended), and an operating system capable of running AMUSE. Linux, Windows, Mac OS X are currently supported.

4 Use Cases

The use cases describe sample usage scenarios for the GPUNIT software, the commands available to the user in each scenario, and the actions taken by GPUNIT in response to each user command. The following table presents the symbols and terminology employed in the use case descriptions and diagrams, along with their definitions.

Term or Symbol	Definition
Action	A command, task, or state change performed by an actor during the scenario.
Actor	A participant in the scenario.
Precondition	A constraint on the state of the world at the start of the scenario. Each constraint is understood to have been fulfilled before the scenario proceeds.
Postcondition	A constraint on the state of the world at the conclusion of the scenario.
●	The point at which the scenario begins and all preconditions must have been fulfilled.
⟶	A transition by which the scenario may proceed from the action or state at the tail of the arrow to the action or state at the arrowhead.
< description >	An action taken by an actor during the scenario. The inner text briefly describes the action.
< description >	A state change that occurs during the scenario as a result of some action. The inner text briefly describes the new state of the world at the particular moment in the scenario.
◉	The point at which the scenario ends and all postconditions must be fulfilled.

4.1 Create an Experiment

4.1.1 Description

The user creates a new experiment, or modifies an existing one, using the graphical interface.

4.1.2 Actors

Name	Description
The user	The person wishing to create an experiment.
The interface	The graphical interface of GPUNIT.

4.1.3 Preconditions

The interface is loaded. Optionally, an experiment is saved in a location accessible to the user.

4.1.4 Actions

The user creates a new experiment or loads an existing one from an accessible location. The interface displays the configuration of modules and entities of which the experiment is composed. In addition, the interface displays a list of available modules and entities. The user selects an existing module or entity, or adds one from the list. The user specifies the initial parameters of the selected module or entity, or opts to delete it. The interface displays the updated state of the experiment. The user repeatedly configured modules and entities in this manner. When the user is satisfied with the state of the experiment, he or she saves the experiment to one or more available formats.

4.1.5 Postconditions

The interface displays a graphical representation of the experiment's configuration.

4.1.6 Workflow Diagram

The following diagram illustrates a workflow associated with this use case.
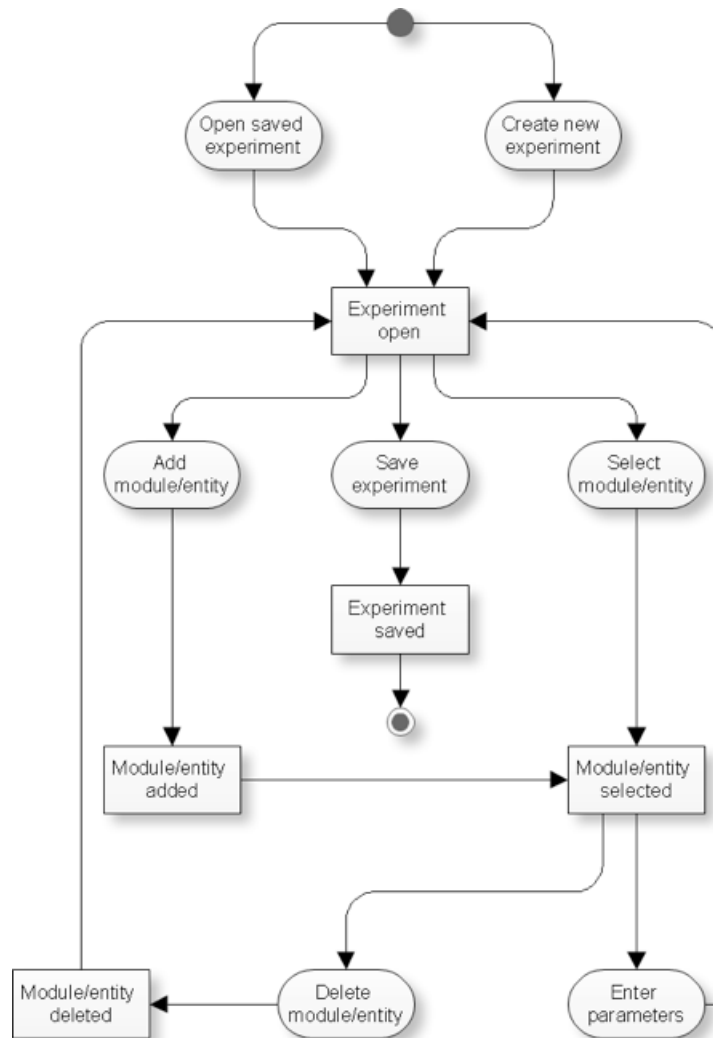
4.2   Run an Experiment Using the Graphical Interface

4.2.1   Description

Using the graphical interface, the user configures and runs an experiment.

4.2.2   Actors

Name	Description
The user	The person wishing to run an experiment.
The interface	The graphical interface of GPUNIT.

4.2.3 Preconditions

An experiment is created and loaded using the interface, as described in 4.1

4.2.4 Actions

The user prepares diagnostics for the experiment, as described in 4.4. The user selects a computing environment in which to run the experiment. The interface generates a simulaton reflecting the experiment's configuration. The interface executes the script using the specified environment, displaying the specified diagnostics. In the event of an error or other unexpected feedback, the interface provides a detailed notification to the user. If the user wishes, he or she may terminate the experiment in progress.

4.2.5 Postconditions

During execution of the experiment, the interface displays diagnostic and logging information. At the conclusion of the experiment, the interface displays the results of the experiment in the format configured by the user.

4.3 Run an Experiment Using the Command-Line Interface

4.3.1 Description

Using the command-line interface, the user runs an experiment.

4.3.2 Actors

Name	Description
The user	The person wishing to run an experiment and possessing sufficiently advanced computing skills to use the command-line interface.
The interface	The graphical interface of GPUNIT.
The system	The GPUnit software package

4.3.3 Preconditions

An experiment is created and saved in a location accessible to the user. The user locates the saved experiment.

4.3.4 Actions

The user provides the interface with the location of the saved experiment and the identity of the computing environment in which to run the simulation. The system validates the location and contents of the saved experiment. The system executes the experiment using the specified environment, displaying diagnostic and logging information. In the event of an error or other unexpected feedback, the interface

provides a detailed notification to the user. If the user wishes, he or she may terminate the experiment in progress.

4.3.5 Postconditions

The interface saves the logs and diagnostics generated while running the experiment for future reference. At the conclusion of the experiment, the interface saves the results of the experiment and notifies the user.

4.4 Prepare Experiment Diagnostics

4.4.1 Description

Using the graphical user interface the user selects diagnostics to be run on the results of the experiment.

4.4.2 Actors

Name	Description
The user	The person wishing to run an experiment.
The interface	The graphical interface of GPUNIT.

4.4.3 Preconditions

The interface is loaded. The user creates a new experiment or loads a saved experiment as described in 4.1.

4.4.4 Actions

The user elects to prepare diagnostics for the experiment. The interface displays a list of available diagnostic scripts. The user selects one or more diagnostic scripts to run alongside the experiment. Upon running the experiment as described in 4.2 or 4.3, the system performs the specified diagnostics.

4.4.5 Postconditions

Diagnostic information pertaining to the experiment is saved in the specified format. If the experiment has been run using the graphical interface as described in 4.3, the interface displays the diagnostic information using the representation selected by the user.

4.5 Advanced User Writes Diagnostics on Experiment

4.5.1 Description

An advanced user uses GPUNIT to create a diagnostics script to be used in later experiments.

4.5.2 Actors

Name	Description
The user	The person wishing to run an experiment and possessing sufficiently advanced computing skills to use the command-line interface.
The interface	The graphical interface of GPUNIT.

4.5.3 Preconditions

GPUNIT is open with an experiment loaded.

4.5.4 Actions

User selects option to create new diagnostics tool. User codes following all python constructs and following the template guidelines of inputs and outputs of diagnostics tool. User saves diagnostics tool.

4.5.5 Postconditions

Diagnostics tool exists to be used for future experiments. Diagnostics tool works assuming all GPUNIT guidelines are followed.

4.6 Use Case Diagram

The following diagram graphically represents the actors involved in each use case and the relationships between the use cases.
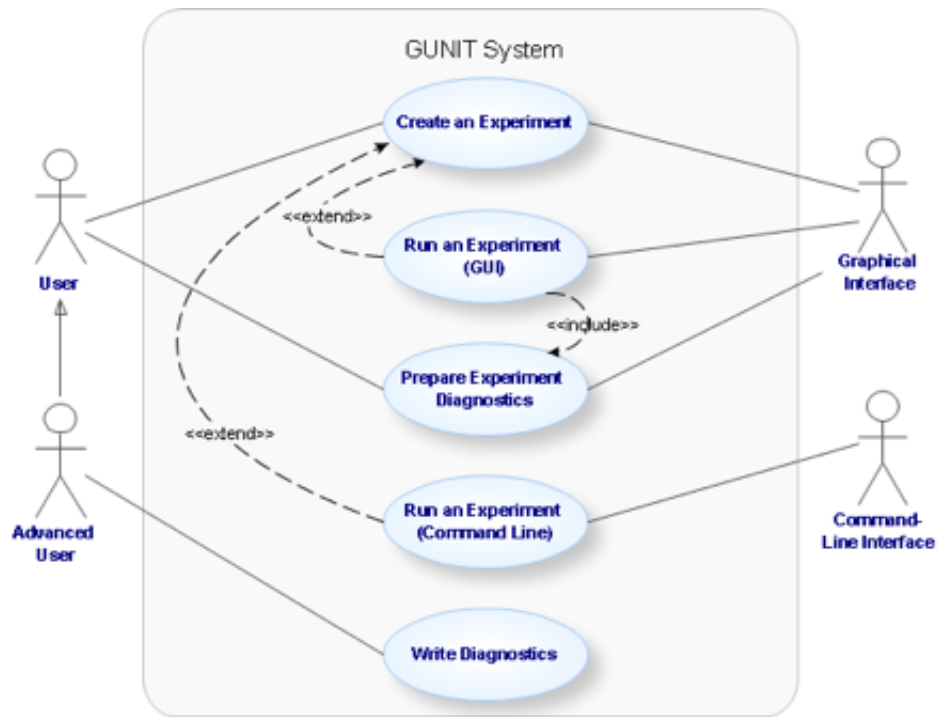
Figure 1: Use Cases

5   User Interface Prototype
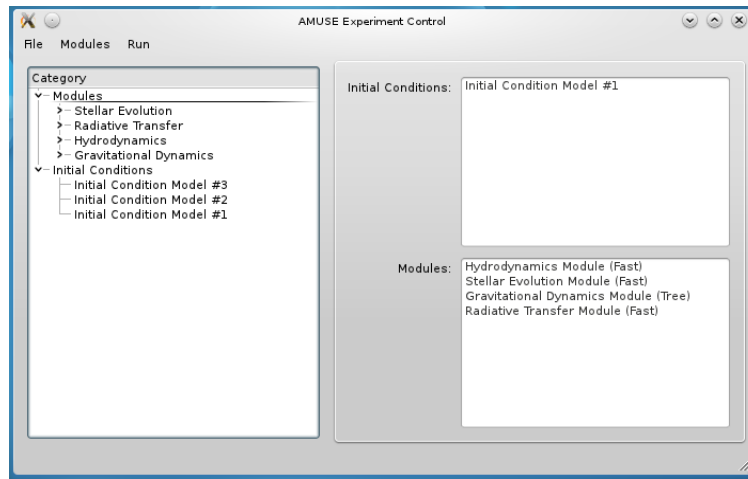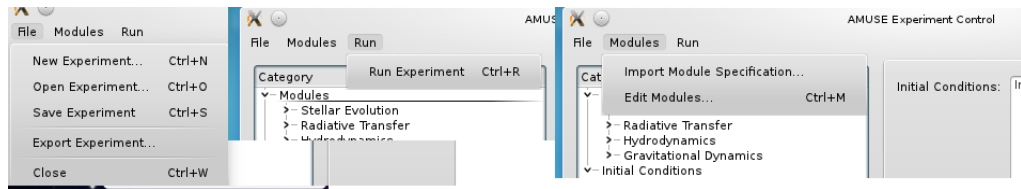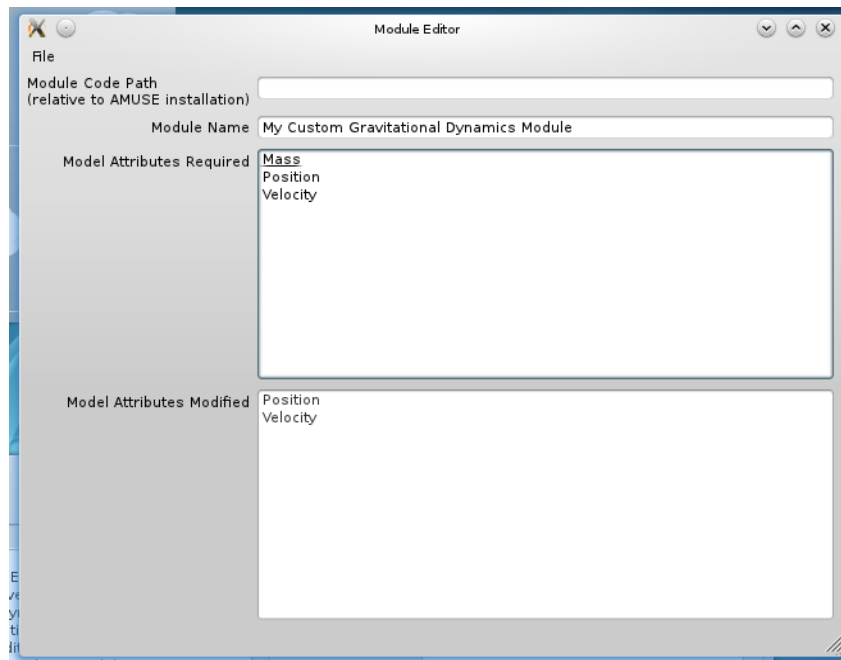
List of Figures

Figure 2: Main View
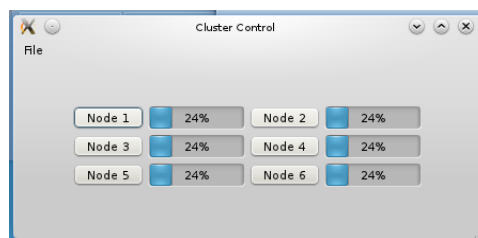

Figure 3: Menus

Figure 4: Module Description Editor

Figure 5: Node View