CS 673
Assignment #6
Software Definition Document (SDD)

Team Snow Crash
Dale Earnest
Jeff Dunn
Dong Luo
Mike McWilliams

Table of Contents

Revision History

Name	Date	Reason For Changes	Version(s)
Dale	07 Nov	Initial Draft	0.1
Dale	15 Nov	Sections 1 and 2 completed	0.2
Dale	16 Nov	Added diagrams from team members, revisions	0.3
Dale	17 Nov	Added additional diagrams from team members	0.4
Dale	19 Nov	Further edits throughout, especially section 2	0.5
Dale	19 Nov	Final version	1.0

1.Introduction

The project is an application that allows the user to create or use default critters and see how they might interact in a simulated world.

This document contains all the design and architecture information necessary to develop the Evolution Simulation desktop application in Java.

This document's audience are the software developers who would develop the system and the business stakeholders that should know the scope of the requirements.

1.1 Design Overview

The "Evolution Simulation" application consists of a user interface, simulation engine, and file import/export subsystem. The user creates and configures custom critters, starting from default templates, and determines the size of the simulated world. Critters are placed in the world according to the user's specifications when the simulation is started. Following the simulation, the user is presented with the results of the simulation.

During critter-configuration time, the user can allot scores from a points pool to each of various Traits. These Traits determine how the critter interacts with the world and other critters. At any time during this phase, the user may choose to export or import critters to or from files.

When the user is satisfied with his critters, he can run the simulation. The simulation may be paused and resumed, or stopped completely, resetting the turns to zero. The simulation has elements of randomness, so the same inputs may not necessarily generate the same outputs. The user may save the state of the world and its critters during a simulation and open that simulation at a later date.

When the simulation is over, the user is presented with a summary of results. The user may save simulation results and open them at a later date. Analysis of results is left to the user.

1.2 Definitions, Acronyms and Abbreviations

Critters – Instances of a Critter Template that populate the World.

Critter Prototypes – The different base types of critters; there are Plants, Prey, and Predators. The prototype determines how the Critter will interact with the world.

Critter Templates – The model of a Critter that the user may design that contains ranges of Trait values that determine the specific Critters that will populate the world. Each template is defined by a Critter Prototype.

GUI- Graphical User Interface, this describes the user interface based on graphics (icons and pictures and menus) instead of text and uses a mouse as well as a keyboard as an input device.

States – These describe the various states that a critter may be in. A state describes an activity that the critter is doing on the world map and may lead to another state.

Traits – A pair of integer values, assigned by the user, that determine how good a critter is at a given task. This is a measure of how successful the critter will be in the world.

Turn – The resolved critter activities in one pass through of the grid. This should take exactly one second to conclude.

World – A bounded grid upon which critters live, interact, and die.

1.3 References

For further information regarding this project, please refer to the following documents:

Software Requirement Specification (SRS)	cs673_hw5_earnest.pdf - this document contains the complete requirements for this prototype.
Understanding Evolution	http://evolution.berkeley.edu/ - This website, sponsored by the University of California at Berkeley, will introduce the reader to basic concepts of genetics as they relate to evolution.
GSON	http://code.google.com/p/google-gson/ - Gson is a Java library that can convert Java Objects into its JSON representative and convert JSON strings into an equivalent Java object.

2. System Architectural Design

2.1 Chosen System Architecture

The “Evolution Simulation” application is a multi-platform desktop application written in Java. It will run on the desktop of any desktop operating system that contains a Java 6 runtime engine (further restrictions detailed in in the constraints section below).

The application uses a variety of design patterns to achieve modularity, extensibility, and cohesion. These include:

- The Model-View-Controller design pattern, or MVC, represents the overall architecture of the application.
    - The View is represented by the GUI and its related classes, contained within the org.snowcrash.gui package and its subpackages.
    - The Controller is represented by the service-layer, which includes several packages – org.snowcrash.configurationservice, org.snowcrash. filemanagement, org.snowcrash.timeengine, and org.snowcrash.world. These packages contain the business logic for how the application behaves.
    - The Model is represented by the DAO and its related classes, contained within the org.snowcrash.dataaccess package.
    - Data structs located in the org.snowcrash.critter (and its subpackages) and org.snowcrash.state are used to pass data between the various components of the system and are stored in the data model.
- The Command design pattern is used to send messages to a component without the client knowing which component is supposed to receive the message.
- The Mediator design pattern is used to further decouple the various aspects of the system by delegating work to the appropriate services and managers.
- The Observer design pattern is used as part of MVC to alert the GUI to changes that occur in the data model.
- The Factory design pattern is used to hide the implementation of some interfaces, such as DAO, Command, and Critter (which is an “interface” in the engineering sense, not the Java sense). The DAO factory enforces the Singleton design pattern.

- The Visitor pattern is used to relay the data that are the results of a command to the client that issued the command. This allows the client to decide how the data will be acted upon without violating the one-way messaging that the commands provide.
    - There is no particular need for commands to be one-way messaging, but it seemed like an appropriate action because very few commands actually needed a return value.
- The State pattern is used to dynamically change critter state at runtime, allowing for maximum flexibility in critter behavior while keeping the complexity isolated in the Critter objects.
- Critter template and world configuration files are stored in the JSON format using the GSON library.

2.2 Assumptions and Constraints

The “Evolution Simulation” application is designed to run on personal computer hardware and software than supports Java 6. Any operating system that can support Java 6 should be able to support this application. This application will run on the Windows XP v5.1 SP3 or later (32-bit), Windows Vista v6.0 SP2 (64-bit), Windows 7 v6.1 (32- and 64-bit), MacOS 10.6 (32- and 64-bit), and Ubuntu Linux v10.04 kernel 2.6.32 (64-bit) operating systems.

We expect the GUI to perform responsibly and to have no appreciable delay between user activities and their results (no longer than .25 to .5 seconds delay). For any given turn, we expect the UI to resolve it in 1 second's time to provide a constant screen update between turns while a simulation is running.

2.3 Software Dependencies

The “Evolution Simulation” application depends on the user previously having installed the Java 6 runtime engine according to their operating systems instructions.

The software utilizes one third party library not included in the Java 6 runtime engine: json-1.5.jar. This library provides the application with the ability to read and write files in the JSON format.

2.4 Alternative Architectures

The design team briefly considered using a web-based 2-tier architecture (database and application server) but the team did not have enough expertise to fairly divide the work between team members. This was also planned around an MVC-style design pattern (using Spring or STRUTS 2 as the web framework).

3.Detailed Description of Components

The following is a detailed description of the components of the application.

3.1 Graphical User Interface

3.1.1Package Summaries

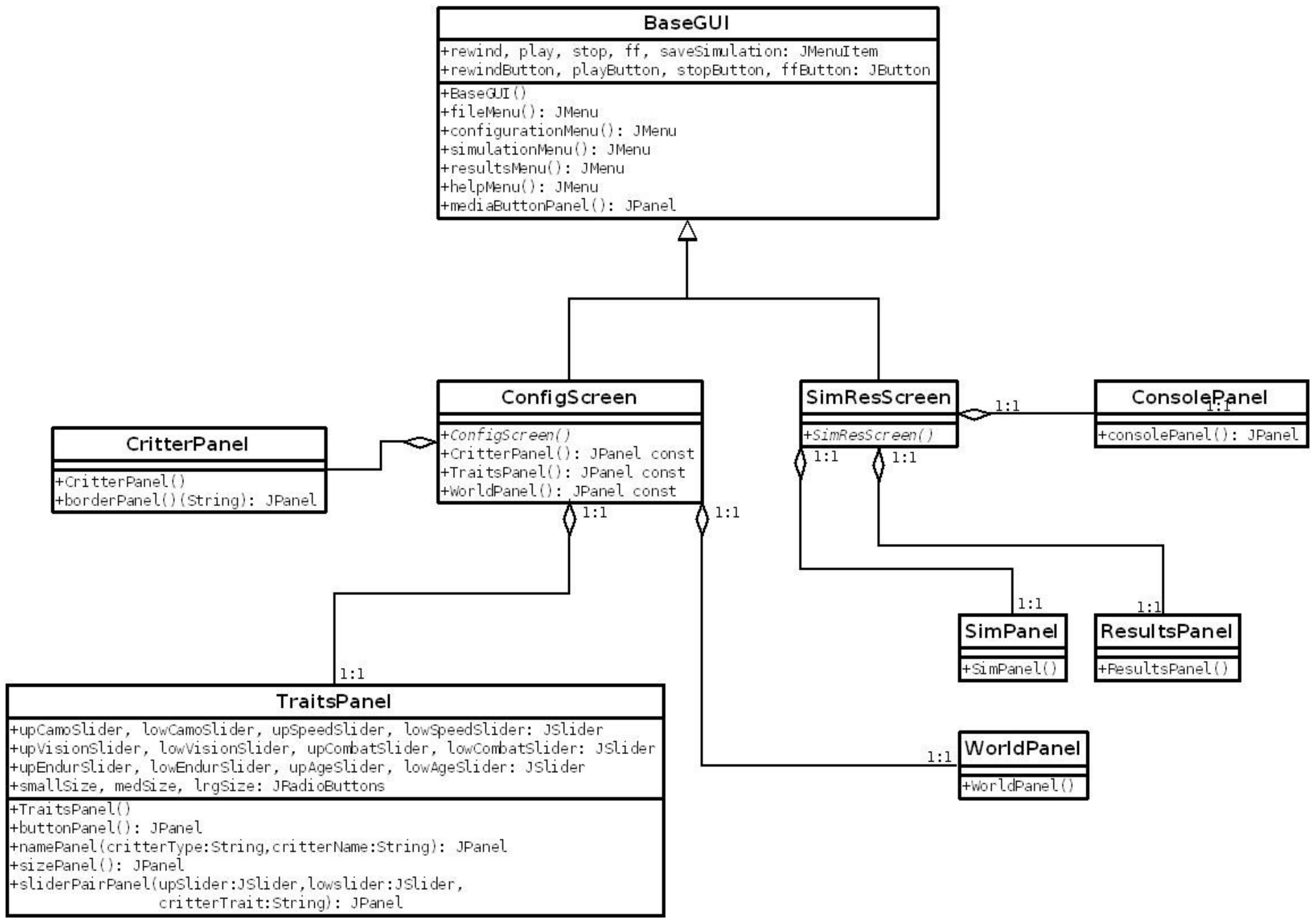
- `org.snowcrash.gui`

SimuPanel	GUI shows the simulation progress.
ResultPanel	GUI shows the simulation results. It is an aggregate for SimResScreen.
ConsolePanel	GUI shows the simulation events in txt format. It is an aggregate for SimResScreen.
ConfigScreen	GUI shows all panels for world and critter configuration. It is a subclass of BaseGUI.
CritterPanel	GUI list of critters. It is an aggregate for ConfigScreen.
TraitsPanel	GUI description of critter traits. It is an aggregate for ConfigScreen.
WorldSettings	GUI description of the simulation world. It is an aggregate for ConfigScreen.
SimResScreen	GUI shows all panels for the simulation, results and console. It is a subclass of BaseGUI.
BaseGUI	GUI that displays all menu items and the media panel. It is the base class for the GUI

- `org.snowcrash.gui.widgets`

CritterTemplateWidget	Facade that provides the functionality of a labeled text box with predetermined dimensions.
SimulationProgressBar	Facade that provides the functionality of a progress bar.

3.1.2 GUI Package Diagram


BaseGUI
+rewind, play, stop, ff, saveSimulation: JMenuItem
+rewindButton, playButton, stopButton, ffButton: JButton
+BaseGUI()
+fileMenu(): JMenu
+configurationMenu(): JMenu
+simulationMenu(): JMenu
+resultsMenu(): JMenu
+helpMenu(): JMenu
+mediaButtonPanel(): JPanel
CritterPanel
+CritterPanel()
+borderPanel()(String): JPanel
ConfigScreen
+ConfigScreen()
+CritterPanel(): JPanel const
+TraitsPanel(): JPanel const
+WorldPanel(): JPanel const
SimResScreen
+SimResScreen()
ConsolePanel
+consolePanel(): JPanel
SimPanel
+SimPanel()
ResultsPanel
+ResultsPanel()
WorldPanel
+WorldPanel()
TraitsPanel
+upCamoSlider, lowCamoSlider, upSpeedSlider, lowSpeedSlider: JSlider
+upVisionSlider, lowVisionSlider, upCombatSlider, lowCombatSlider: JSlider
+upEndurSlider, lowEndurSlider, upAgeSlider, lowAgeSlider: JSlider
+smallSize, medSize, lrgSize: JRadioButtons
+TraitsPanel()
+buttonPanel(): JPanel
+namePanel(critterType:String,critterName:String): JPanel
+sizePanel(): JPanel
+sliderPairPanel(upSlider:JSlider,lowslider:JSlider,
critterTrait:String): JPanel
1:1

3.2 Controller Layer

3.2.1Package Summaries

- `org.snowcrash.commands`

AbstractCommand	Base class for all commands. Implements the Command interface so child classes do not have to.
CallbackCommand	Abstract subclass of AbstractCommand that allows data to be passed back to the caller.
Command	Super-interface for all commands.
CommandFactory	Provides a means for classes outside the commands package to generate commands.
CommandMediator	Serves as a mediator for the various parts of the program; handles commands by calling the correct components.
CreateCritterTemplateCommand	Command to create a critter template.
DeleteCritterTemplateCommand	Command to delete a critter template.
ExportCritterTemplatesCommand	Command to export all current critter templates.
FileCommand	Abstract subclass of AbstractCommand that works with file i/o.
GetCritterTemplateCommand	Command to retrieve a critter template.
ImportCritterTemplatesCommand	Command to import all critter templates from a file.
LoadConfigurationCommand	Command to load a previous configuration from a file.
ModifyCritterTemplateCommand	Command to modify a critter template.
SaveConfigurationCommand	Command to save the configuration to a file for future use.

- `org.snowcrash.configurationservice`

ConfigurationManager	Implements methods for configuring critter templates.
IConfigurationManager	Exposes methods for configuring critter templates.

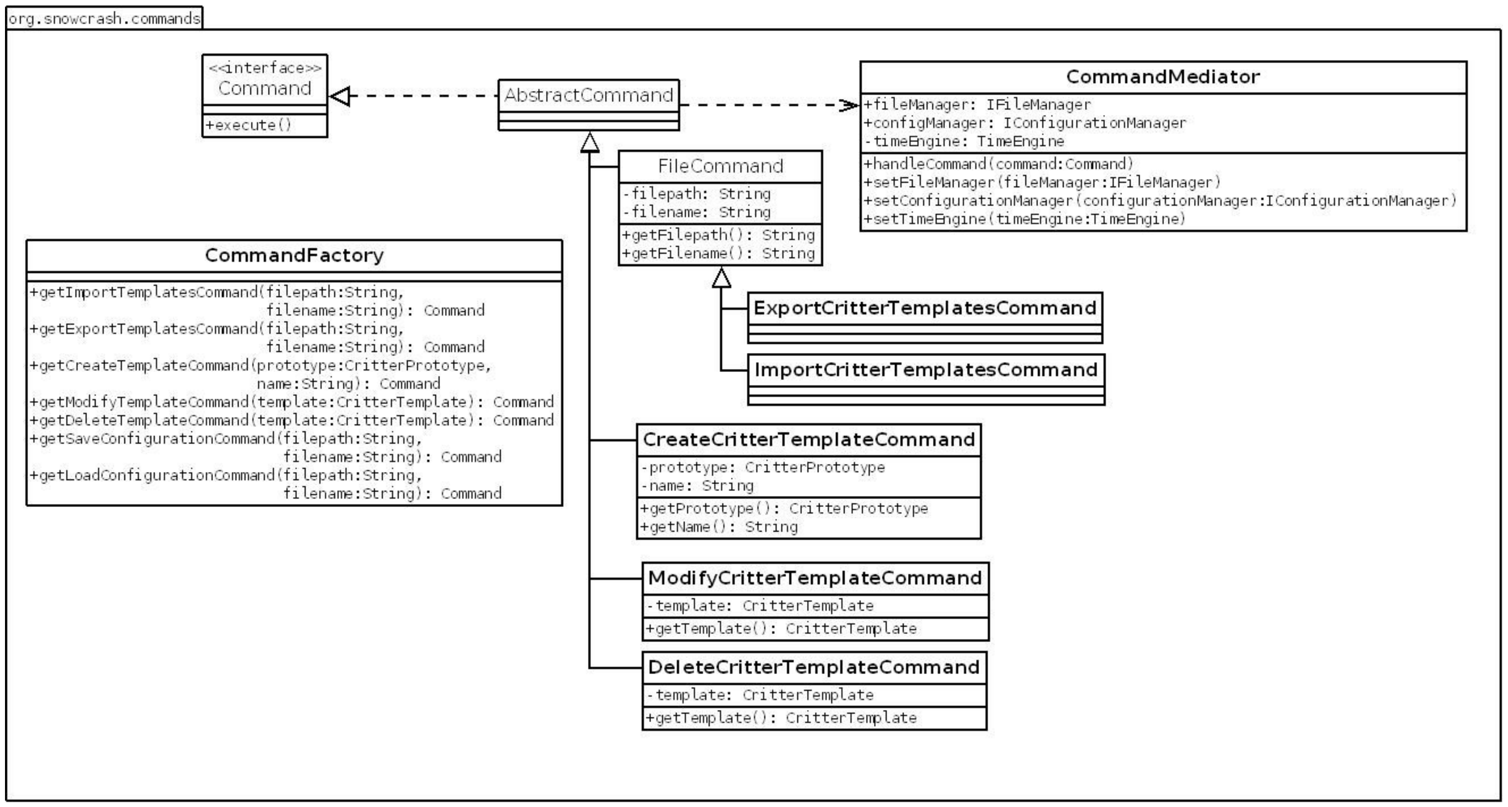
- `org.snowcrash.dataaccess`

CachedDAO	Cache to store data.
CachedTable	A table in the cache database.
DAO	Exposes methods for storing and retrieving data from a database.
DAOException	Subclass of Exception for specifying that an exception occurred during an attempt to store data in the database.
DAOExceptionMessages	Specifies error messages for use by implementing classes.
DAOFactory	Provides a means for classes outside the dataaccess package to obtain a DAO reference. Enforces a singleton implementation of the DAO.
DatabaseObject	Interface for storing an object in the database.
InvalidInputDAOException	Subclass of DAOException that specifies that the exception was caused by invalid input.

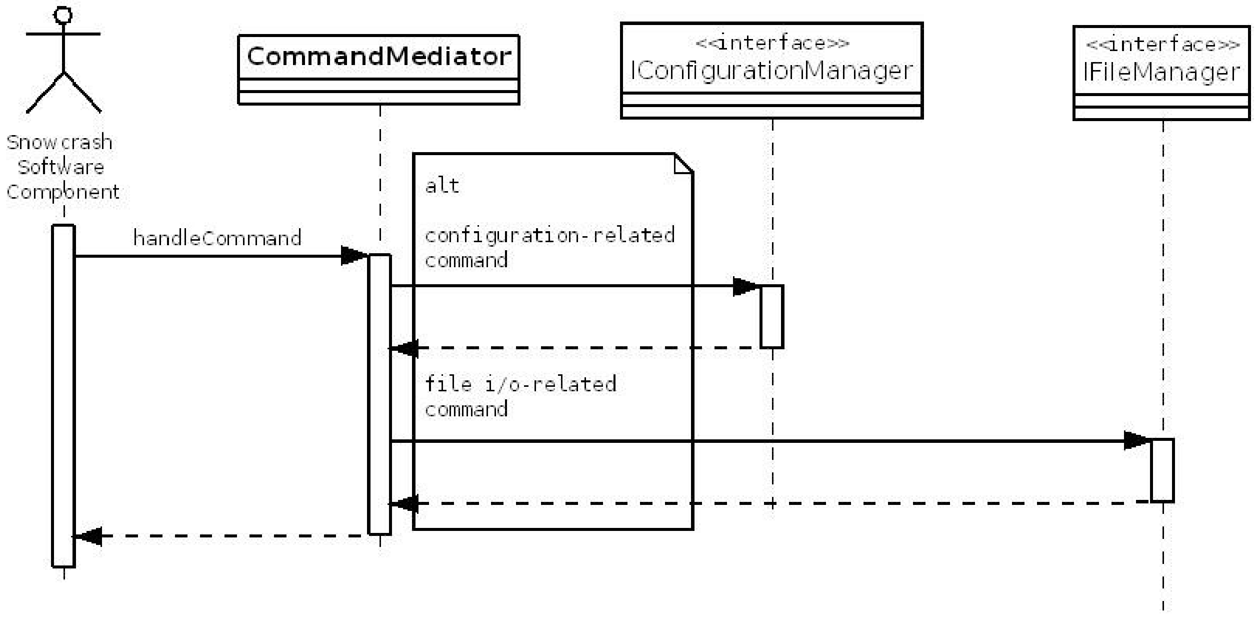
- `org.snowcrash.timeengine`

TimeEngine	Determines how often a “tick” in a simulation occurs. Default is 1000ms.
TimeListener	Defines the interface for listening to time tick events.

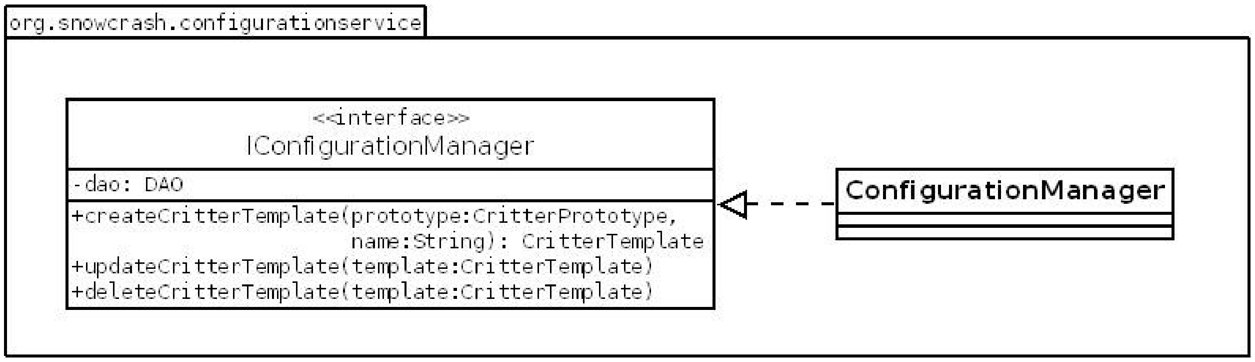
3.2.2Command Package Diagram


org.snowcrash.commands
<<interface>>
Command
+execute()
AbstractCommand
CommandMediator
+fileManager: IFileManager
+configManager: IConfigurationManager
-timeEngine: TimeEngine
+handleCommand(command:Command)
+setFileManager(fileManager:IFileManager)
+setConfigurationManager(configurationManager:IConfigurationManager)
+setTimeEngine(timeEngine:TimeEngine)
FileCommand
-filepath: String
-filename: String
+getFilepath(): String
+getFilename(): String
CommandFactory
+getImportTemplatesCommand(filepath:String, filename:String): Command
+getExportTemplatesCommand(filepath:String, filename:String): Command
+getCreateTemplateCommand(prototype:CritterPrototype, name:String): Command
+getModifyTemplateCommand(template:CritterTemplate): Command
+getDeleteTemplateCommand(template:CritterTemplate): Command
+getSaveConfigurationCommand(filepath:String, filename:String): Command
+getLoadConfigurationCommand(filepath:String, filename:String): Command
ExportCritterTemplatesCommand
ImportCritterTemplatesCommand
CreateCritterTemplateCommand
-prototype: CritterPrototype
-name: String
+getPrototype(): CritterPrototype
+getName(): String
ModifyCritterTemplateCommand
-template: CritterTemplate
+getTemplate(): CritterTemplate
DeleteCritterTemplateCommand
-template: CritterTemplate
+getTemplate(): CritterTemplate

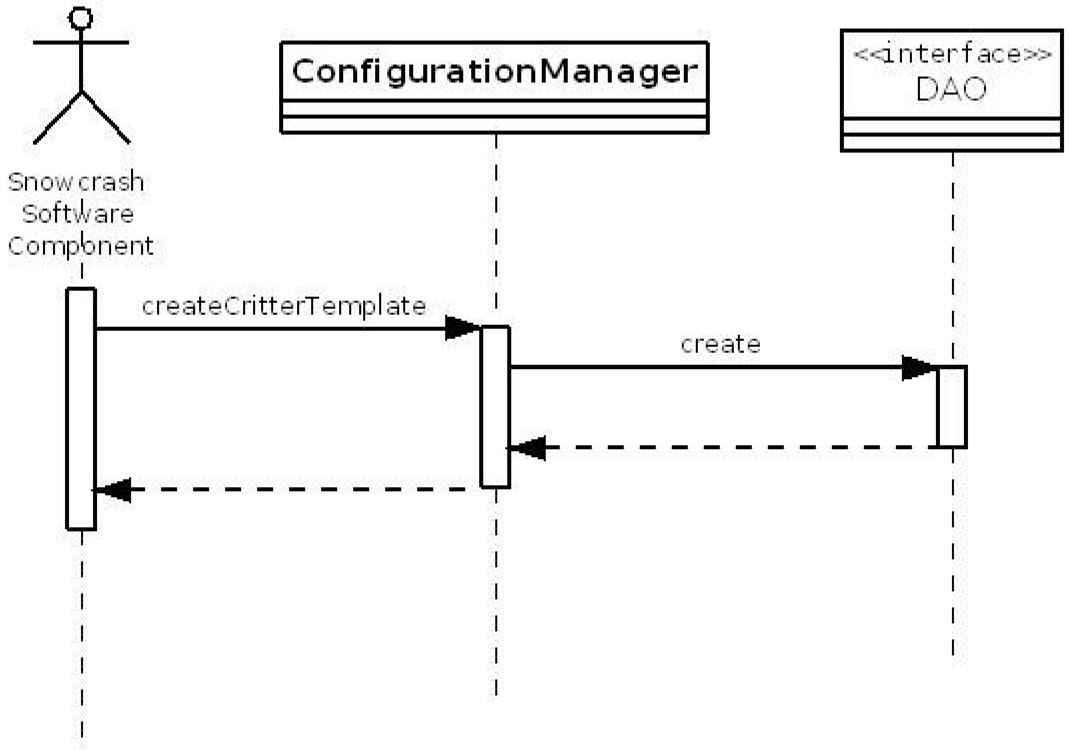
3.2.3Command Sequence Diagrams

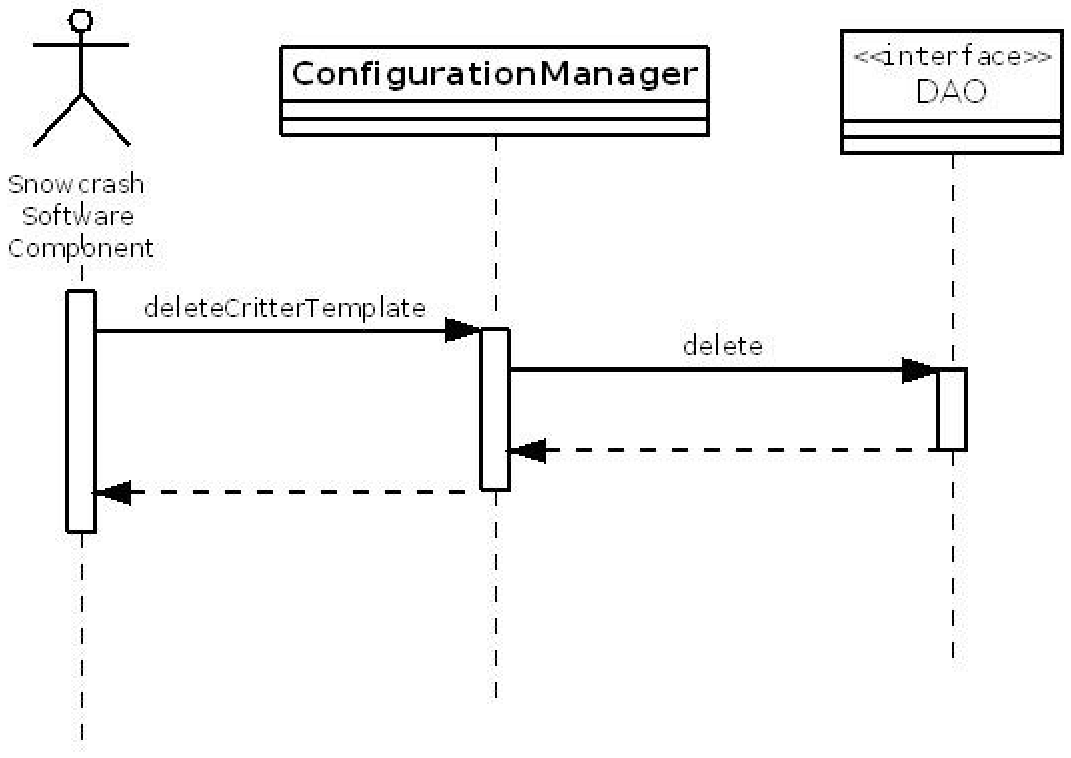


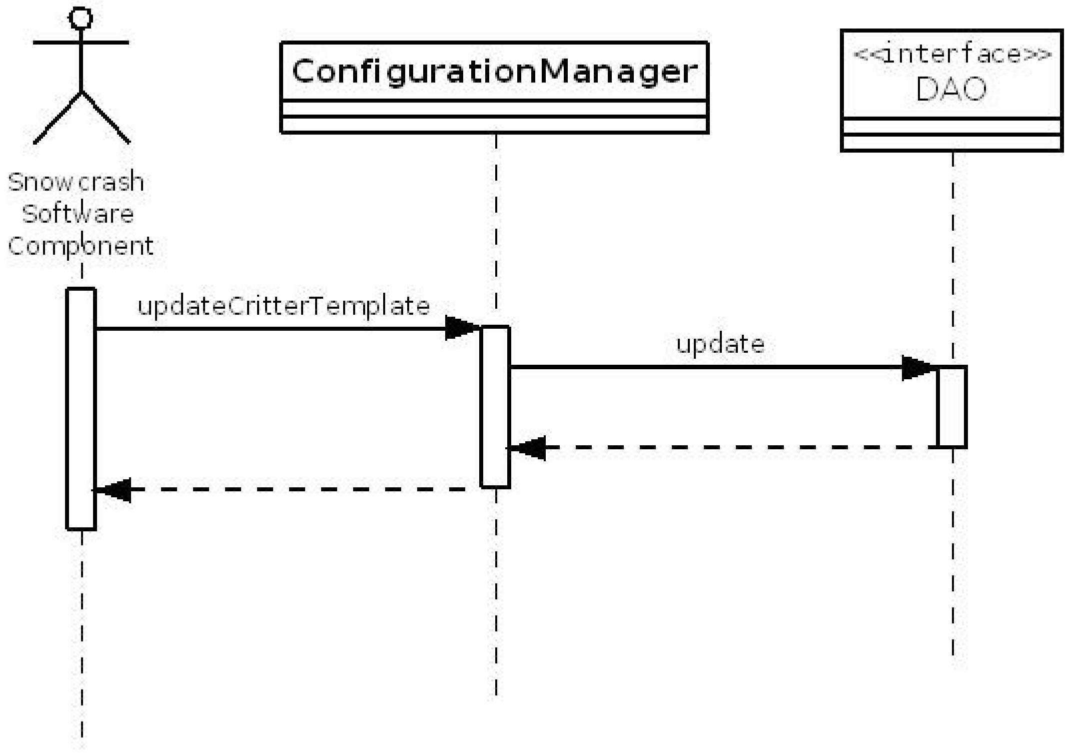
3.2.4Configuration Package Diagram


org.snowcrash.configurationservice
<<interface>>
IConfigurationManager
-dao: DAO
+createCritterTemplate(prototype:CritterPrototype, name:String): CritterTemplate
+updateCritterTemplate(template:CritterTemplate)
+deleteCritterTemplate(template:CritterTemplate)
ConfigurationManager

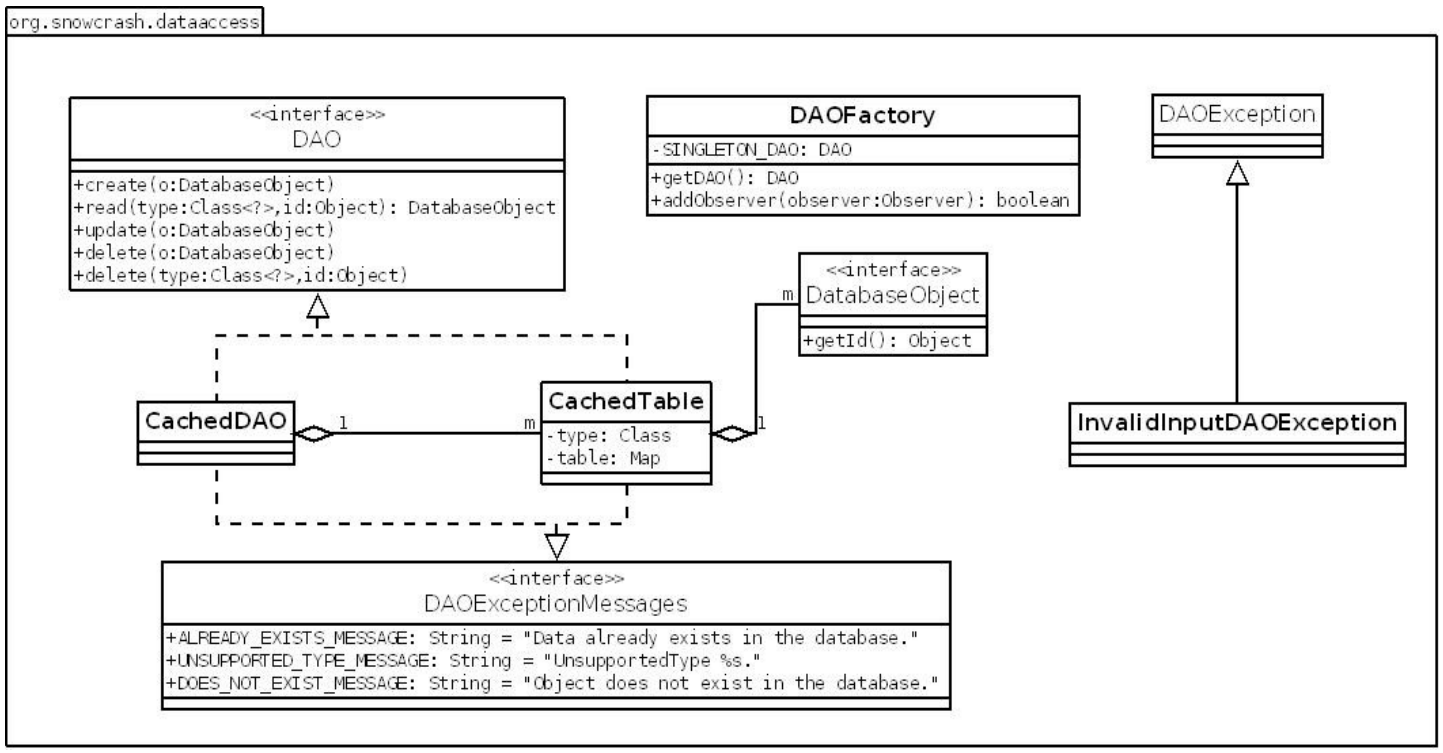
3.2.5Configuration Sequence Diagrams



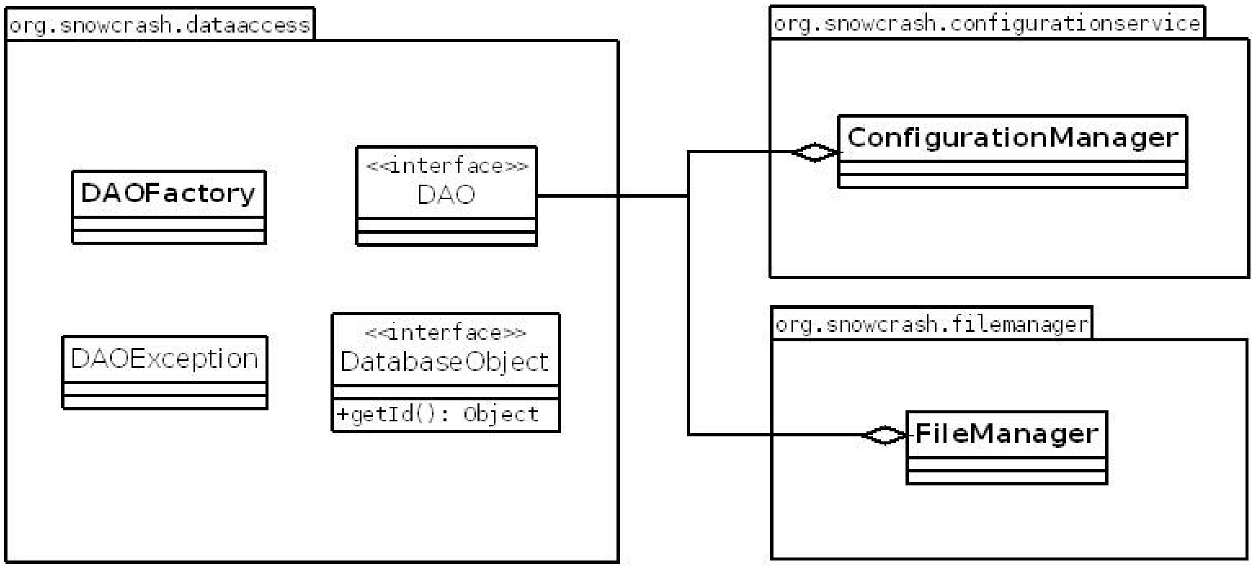

Snowcrash Software Component
ConfigurationManager
<<interface>> DAO
deleteCritterTemplate
delete


Snowcrash Software Component
ConfigurationManager
<<interface>> DAO
updateCritterTemplate
update

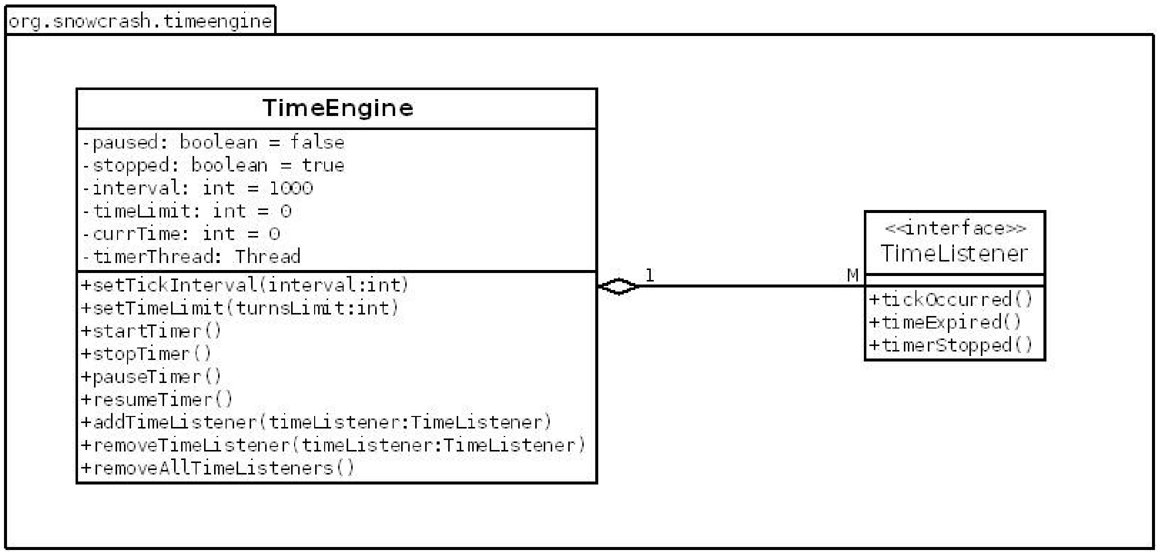
3.2.6Data Access Package Diagram


org.snowcrash.dataaccess
<<interface>>
DAO
+create(o:DatabaseObject)
+read(type:Class<?>,id:Object): DatabaseObject
+update(o:DatabaseObject)
+delete(o:DatabaseObject)
+delete(type:Class<?>,id:Object)
DAOFactory
-SINGLETON_DAO: DAO
+getDAO(): DAO
+addObserver(observer:Observer): boolean
DAOException
<<interface>>
DatabaseObject
+getId(): Object
m
CachedDAO
1
m
CachedTable
-type: Class
-table: Map
1
InvalidInputDAOException
<<interface>>
DAOExceptionMessages
+ALREADY_EXISTS_MESSAGE: String = "Data already exists in the database."
+UNSUPPORTED_TYPE_MESSAGE: String = "UnsupportedType %s."
+DOES_NOT_EXIST_MESSAGE: String = "Object does not exist in the database."

3.2.7Data Access Relationship Diagram



3.2.8Time Engine Package Diagram



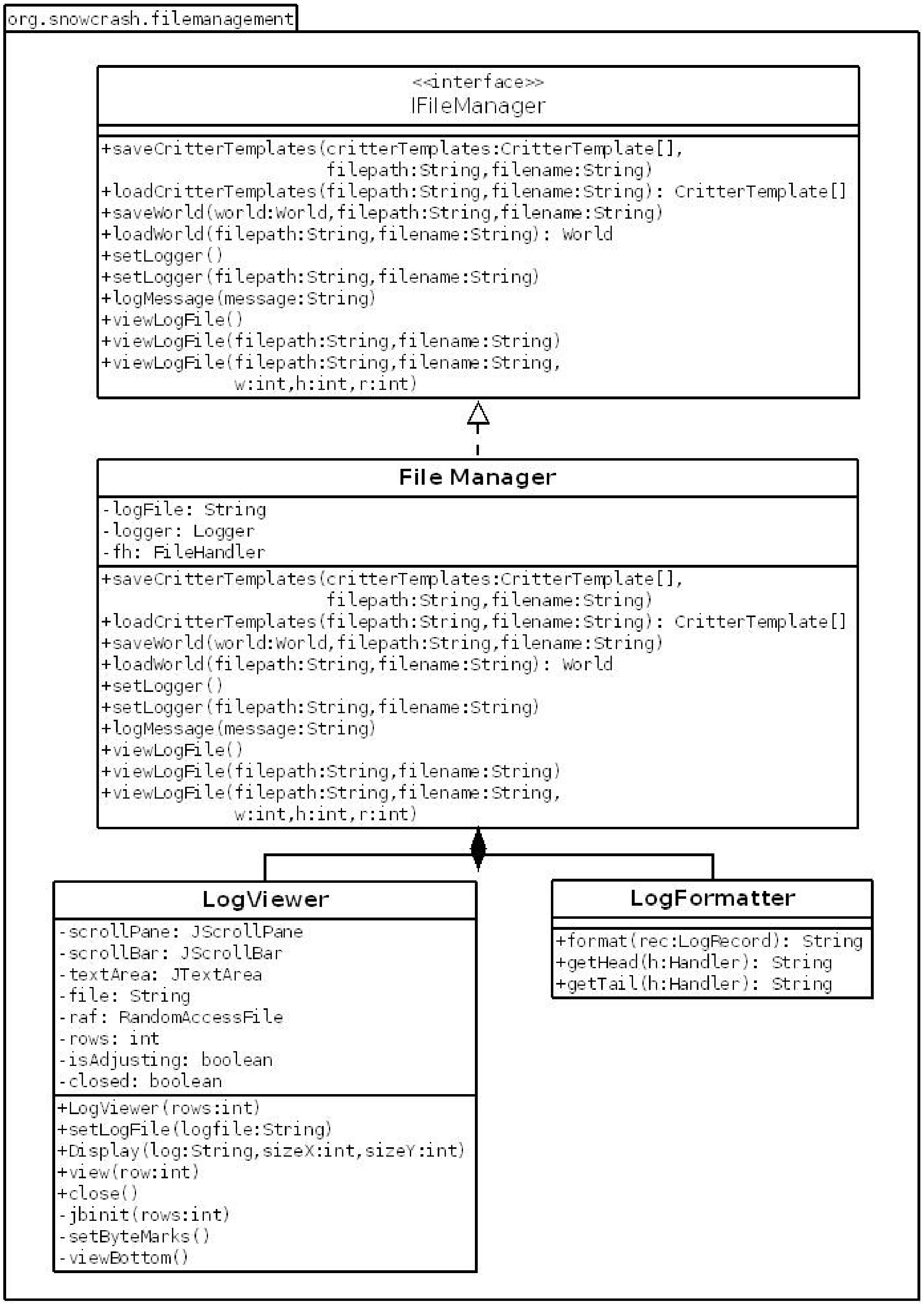
3.3 File I/O Manager

3.3.1Package Summaries

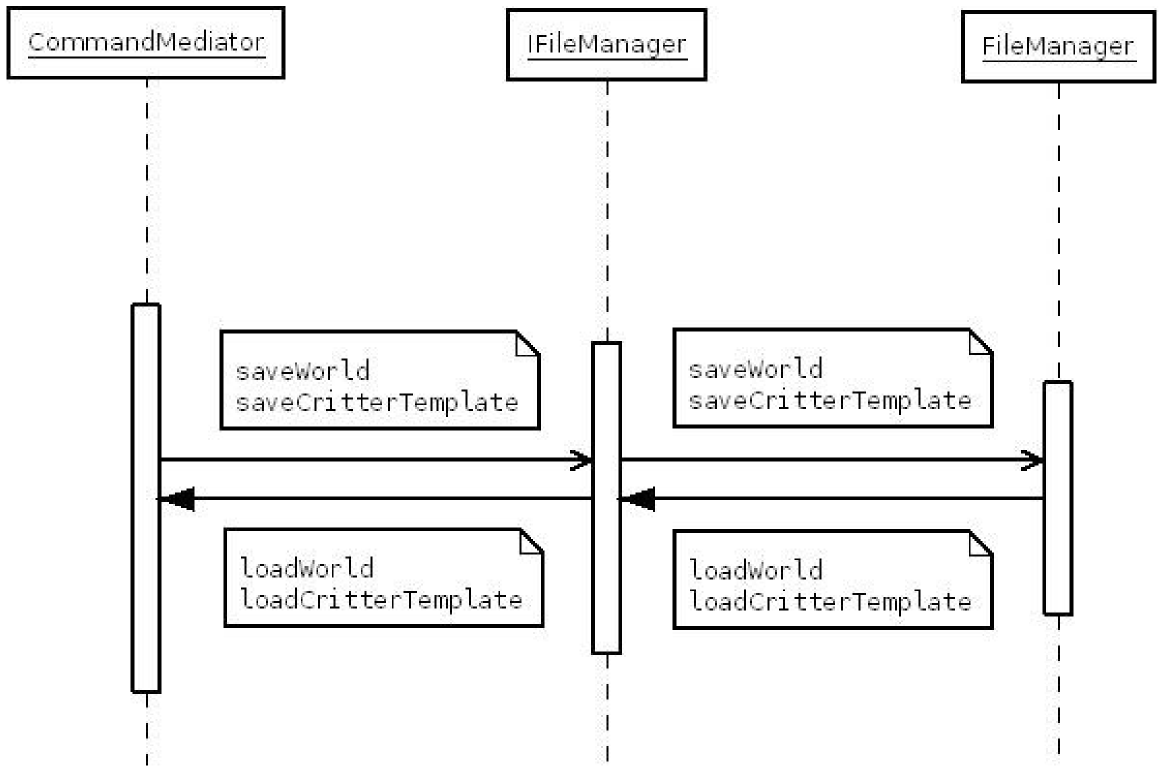
- `org.snowcrash.filemanagement`

IfileManager	Interface that exposes the methods of FileManager class.
FileManager	Implements methods for File I/O.
LogFormatter	Provides customized format of the log message.
LogViewer	Provides a means to display specified log file in a popup window.

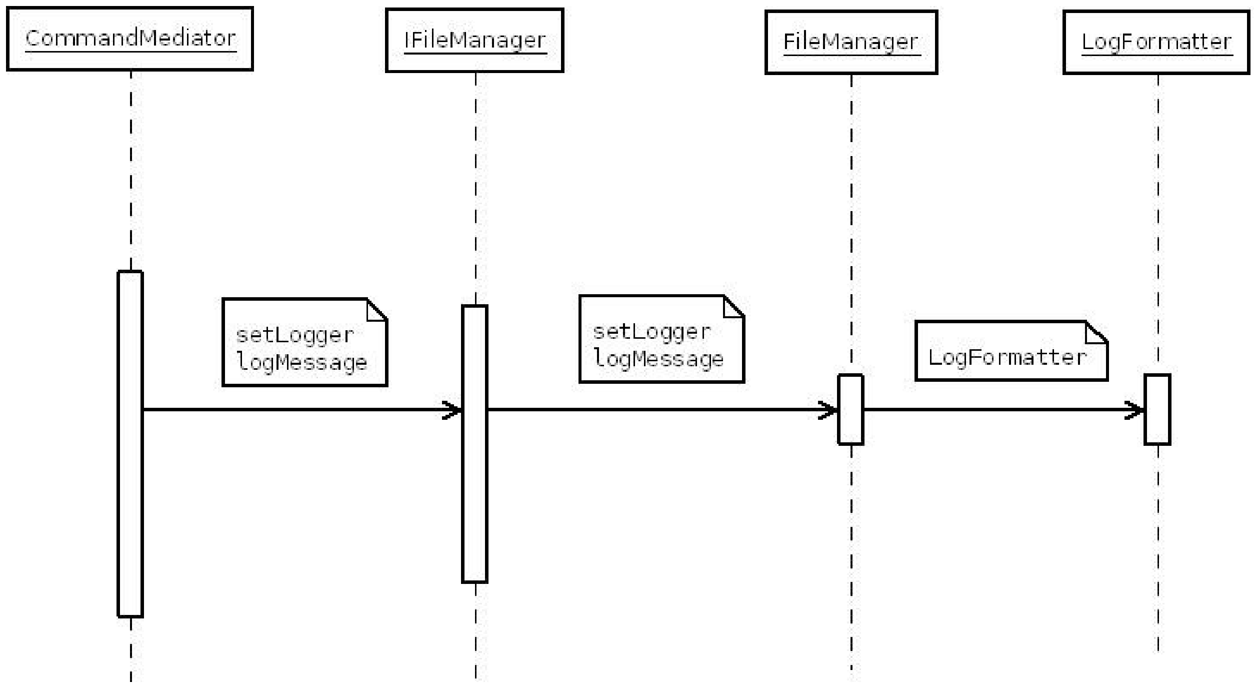
3.3.2File Manager Class Diagram


org.snowcrash.filemanagement
<<interface>>
IFileManager
+saveCritterTemplates(critterTemplates:CritterTemplate[], filepath:String,filename:String)
+loadCritterTemplates(filepath:String,filename:String): CritterTemplate[]
+saveWorld(world:World,filepath:String,filename:String)
+loadWorld(filepath:String,filename:String): World
+setLogger()
+setLogger(filepath:String,filename:String)
+logMessage(message:String)
+viewLogFile()
+viewLogFile(filepath:String,filename:String)
+viewLogFile(filepath:String,filename:String, w:int,h:int,r:int)
File Manager
-logFile: String
-logger: Logger
-fh: FileHandler
+saveCritterTemplates(critterTemplates:CritterTemplate[], filepath:String,filename:String)
+loadCritterTemplates(filepath:String,filename:String): CritterTemplate[]
+saveWorld(world:World,filepath:String,filename:String)
+loadWorld(filepath:String,filename:String): World
+setLogger()
+setLogger(filepath:String,filename:String)
+logMessage(message:String)
+viewLogFile()
+viewLogFile(filepath:String,filename:String)
+viewLogFile(filepath:String,filename:String, w:int,h:int,r:int)
LogViewer
-scrollPane: JScrollPane
-scrollBar: JScrollBar
-textArea: JTextArea
-file: String
-raf: RandomAccessFile
-rows: int
-isAdjusting: boolean
-closed: boolean
+LogViewer(rows:int)
+setLogFile(logfile:String)
+Display(log:String,sizeX:int,sizeY:int)
+view(row:int)
+close()
-jbinit(rows:int)
-setByteMarks()
-viewBottom()
LogFormatter
+format(rec:LogRecord): String
+getHead(h:Handler): String
+getTail(h:Handler): String

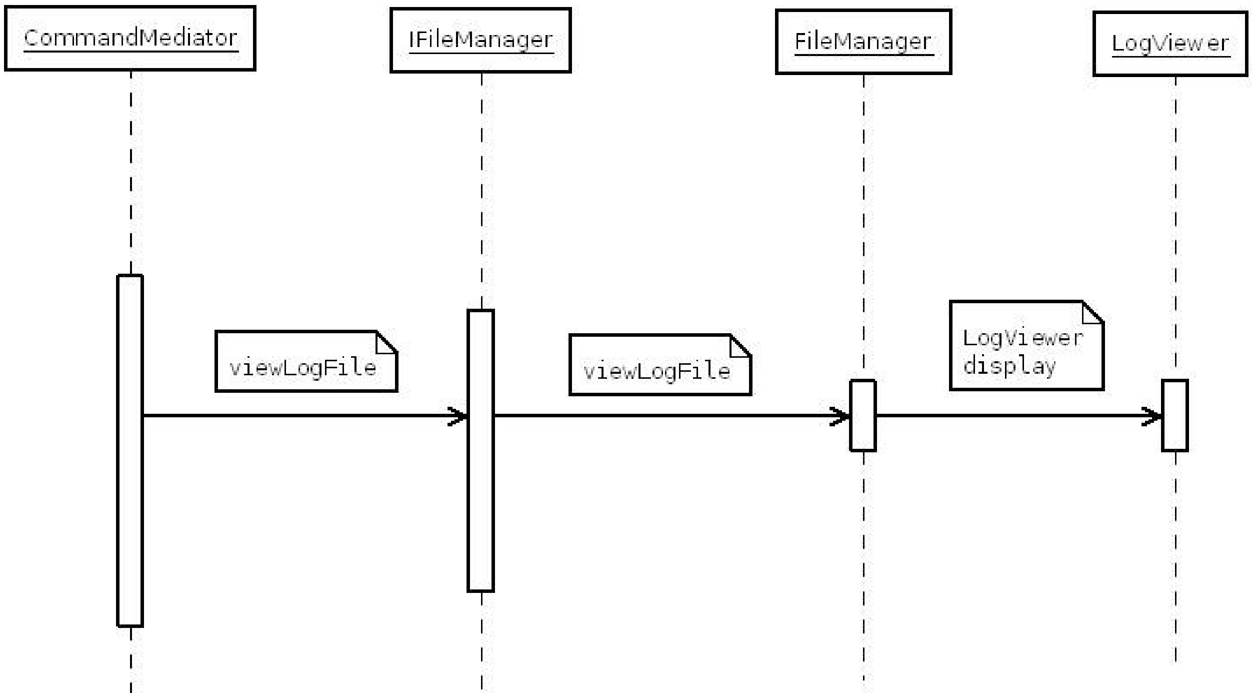
3.3.3File Manage Load/Save Sequence



3.3.4File Logger Sequence



3.3.5File Log Viewer



3.4 World, Critter, and State Model

3.4.1Package Summaries

- `org.snowcrash.critter`

Critter	Model object that stores specific critter instance data.
CritterFactory	Singleton that returns a Critter based on a template or two parents.
CritterTemplate	Parent templates upon which all critter instances are based. Users import and export critter templates for use in the simulation.

- `org.snowcrash.critter.data`

CritterPrototype	Enum that specifies the different critter prototypes.
Size	Enum that specifies the different critter sizes.
Trait	Enum that specifies the different critter traits.

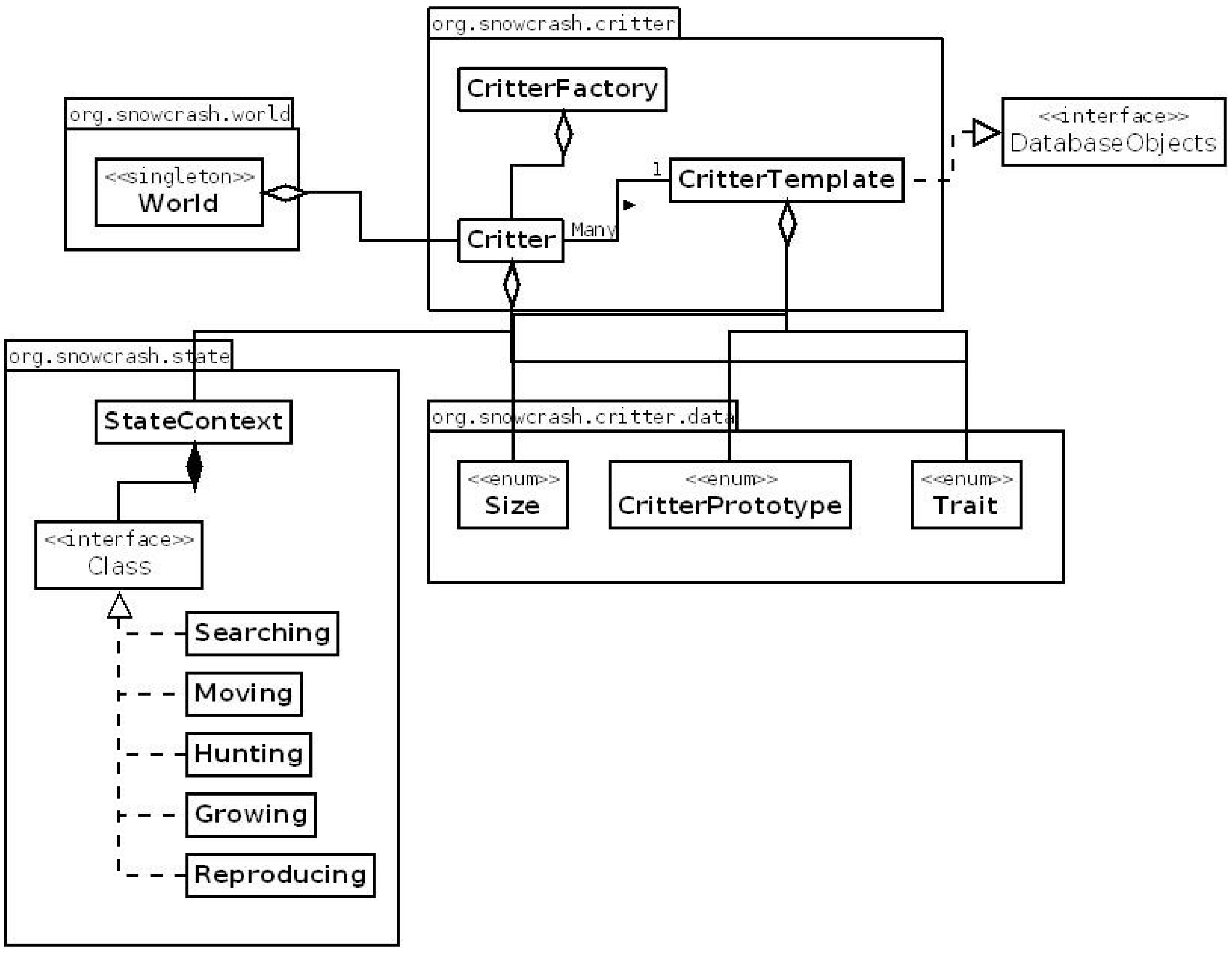
- `org.snowcrash.world`

World	Singleton grid upon which critters live, die, hunt, and reproduce.

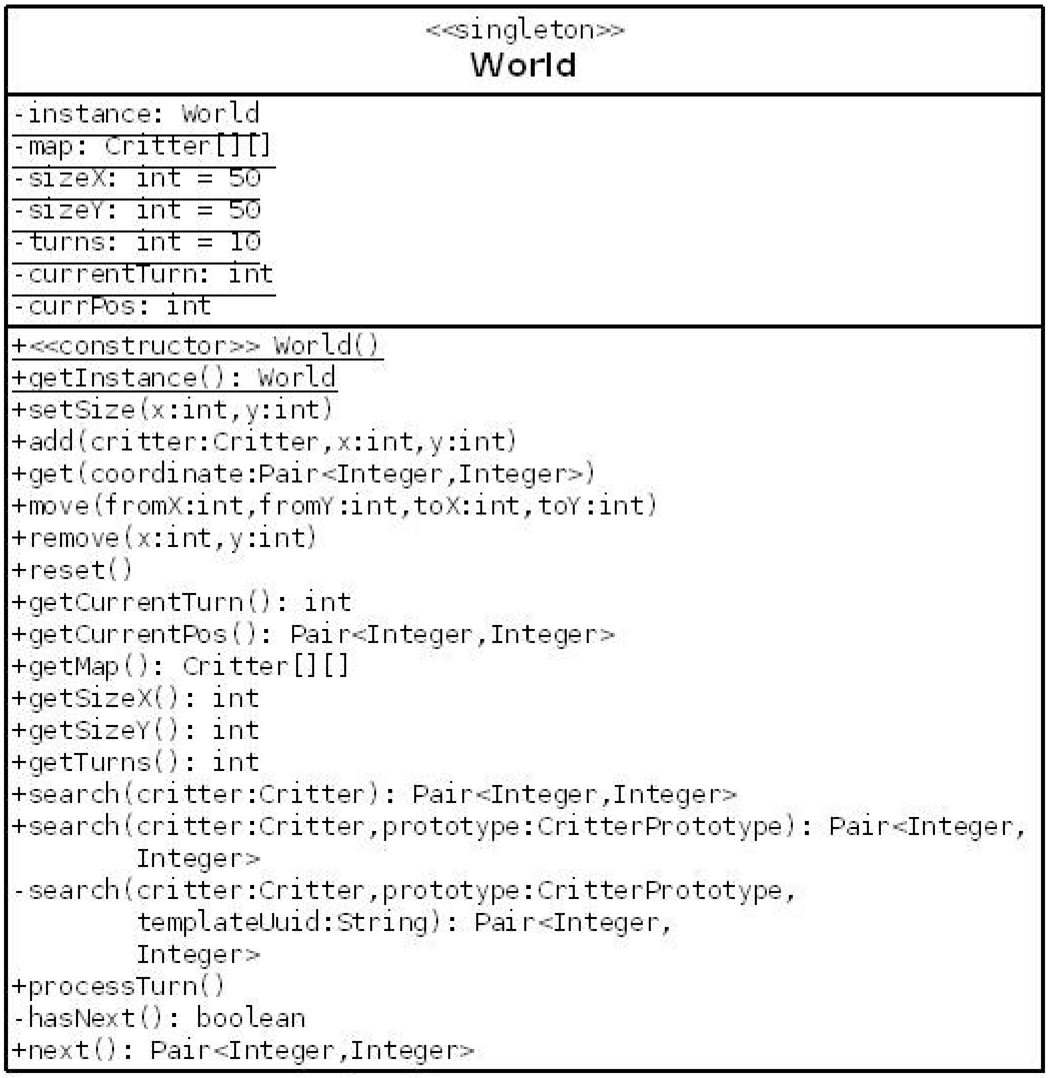
- `org.snowcrash.state`

Growing	Plants regenerate health.
Hunting	Predators and Prey search for food.
Moving	Governs how critters move in the world.
Reproducing	Governs how critters reproduce.
Searching	Determines what actions a critter will take.
State	Interface that all states must conform to.
StateContext	Manages all states and state transitions for a given critter.

3.4.2Relational Model

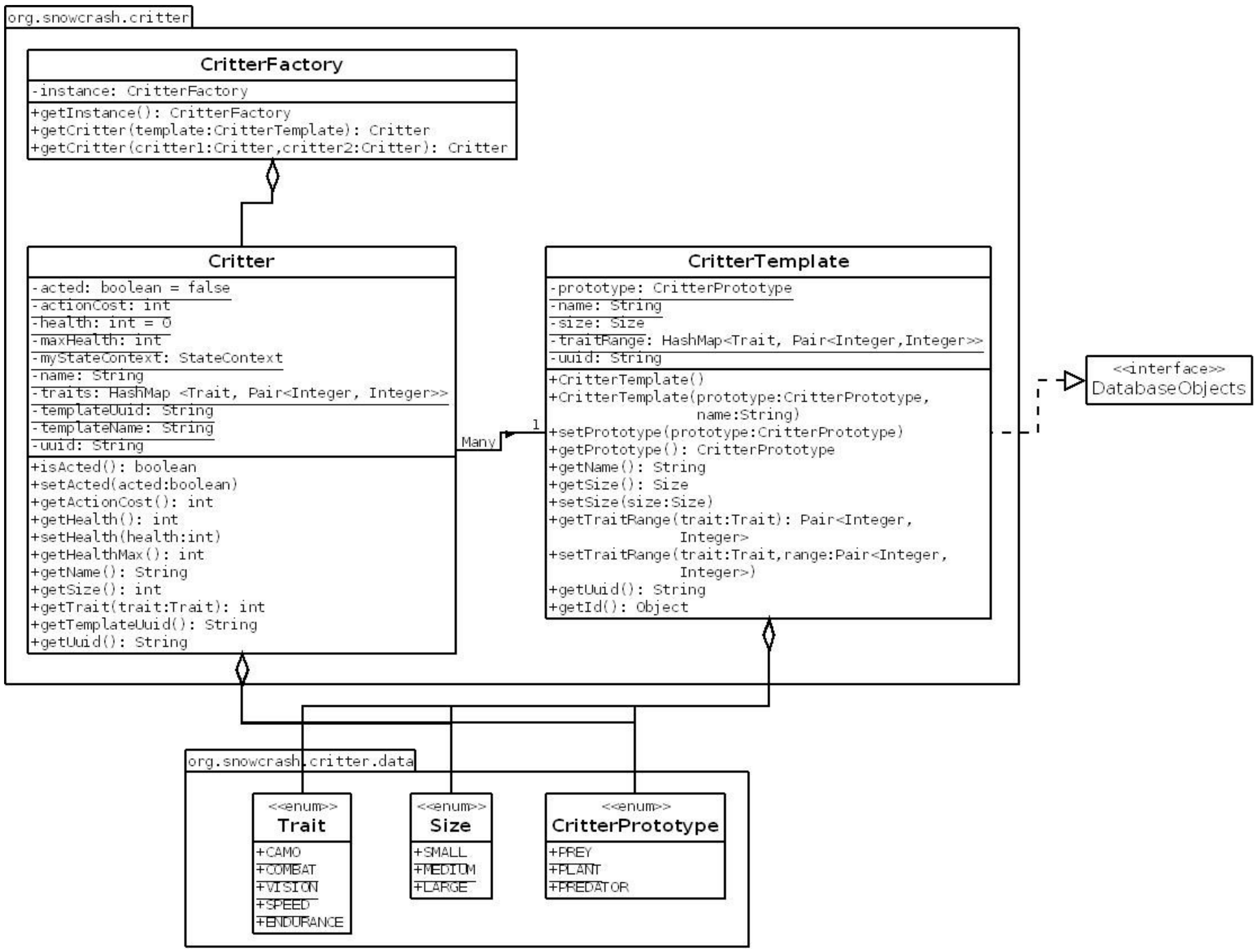


3.4.3World Package Diagram

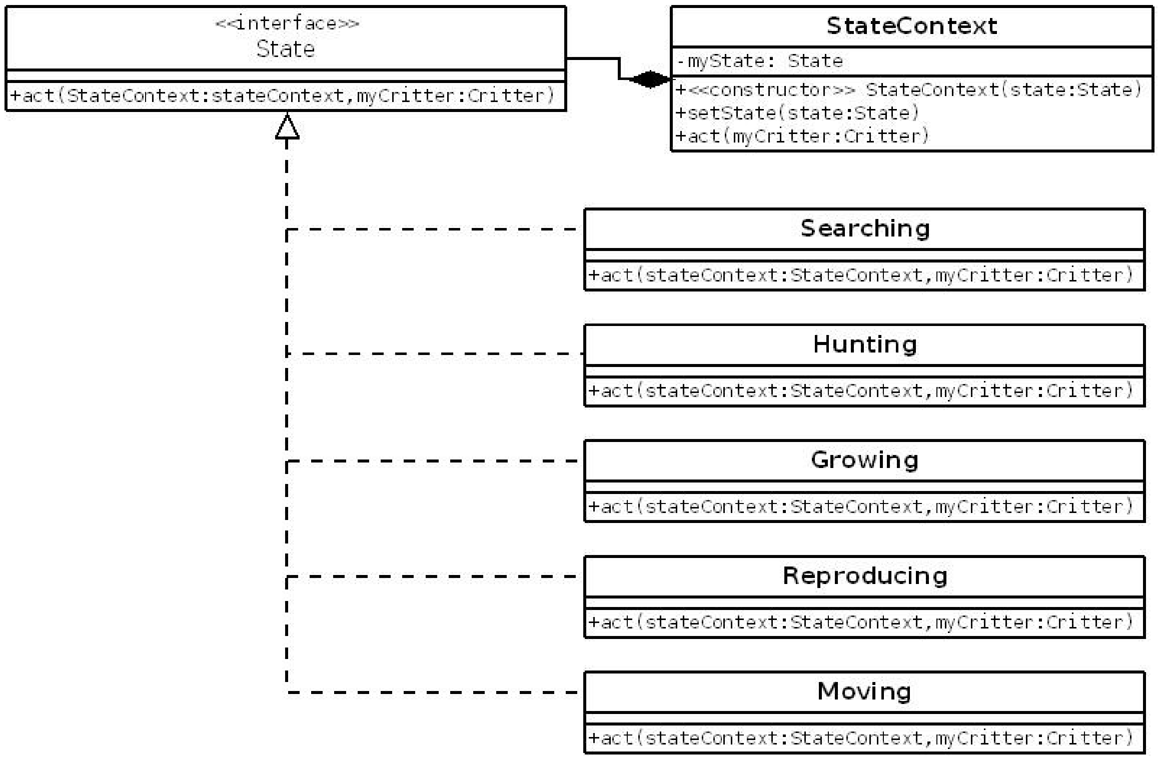


```
-instance: World
-map: Critter[][]
-sizeX: int = 50
-sizeY: int = 50
-turns: int = 10
-currentTurn: int
-currPos: int
+<<constructor>> World()
+getInstance(): World
+setSize(x:int,y:int)
+add(critter:Critter,x:int,y:int)
+get(coordinate:Pair<Integer,Integer>)
+move(fromX:int,fromY:int,toX:int,toY:int)
+remove(x:int,y:int)
+reset()
+getCurrentTurn(): int
+getCurrentPos(): Pair<Integer,Integer>
+getMap(): Critter[][]
+getSizeX(): int
+getSizeY(): int
+getTurns(): int
+search(critter:Critter): Pair<Integer,Integer>
+search(critter:Critter,prototype:CritterPrototype): Pair<Integer,
        Integer>
-search(critter:Critter,prototype:CritterPrototype,
        templateUuid:String): Pair<Integer,
        Integer>
+processTurn()
-hasNext(): boolean
+next(): Pair<Integer,Integer>
```

3.4.4Critter Package Diagram



3.4.5 Critter State Model


<<interface>>
State
+act(StateContext:stateContext,myCritter:Critter)
StateContext
-myState: State
+<<constructor>> StateContext(state:State)
+setState(state:State)
+act(myCritter:Critter)
Searching
+act(stateContext:StateContext,myCritter:Critter)
Hunting
+act(stateContext:StateContext,myCritter:Critter)
Growing
+act(stateContext:StateContext,myCritter:Critter)
Reproducing
+act(stateContext:StateContext,myCritter:Critter)
Moving
+act(stateContext:StateContext,myCritter:Critter)

3.4.6 Searching State Pseudocode

```
• if PLANT
    ○ if HEALTH > 75%
        ▪ Change State to Reproducing
    ○ else
        ▪ Change State to Growing
• if PREY or PREDATOR
    ○ if HEALTH > 75%
        ▪ Change State to Reproducing
    ○ else
        ▪ Change State to Hunting
```

3.4.7Hunting State Pseudocode

```
• if candidate victims in the world found [hunter VISION – victim CAMO + hunter SIZE
  – victim SIZE + random#]
    ○ if attack successful [(hunter VISION – victim VISION) / hunter SPEED + hunter
      SPEED * distance / (hunter SPEED – victim SPEED) + hunter COMBAT – victim
      COMBAT < hunter ENDURANCE]
        ▪ victim dies
        ▪ hunter fills up health
        ▪ hunter replaces victim on grid
    ○ else
```

```
            ▪ Change State to Moving
```

3.4.8Growing State Pseudocode

```
• Regenerate health by HEALTH_REGEN value.
```

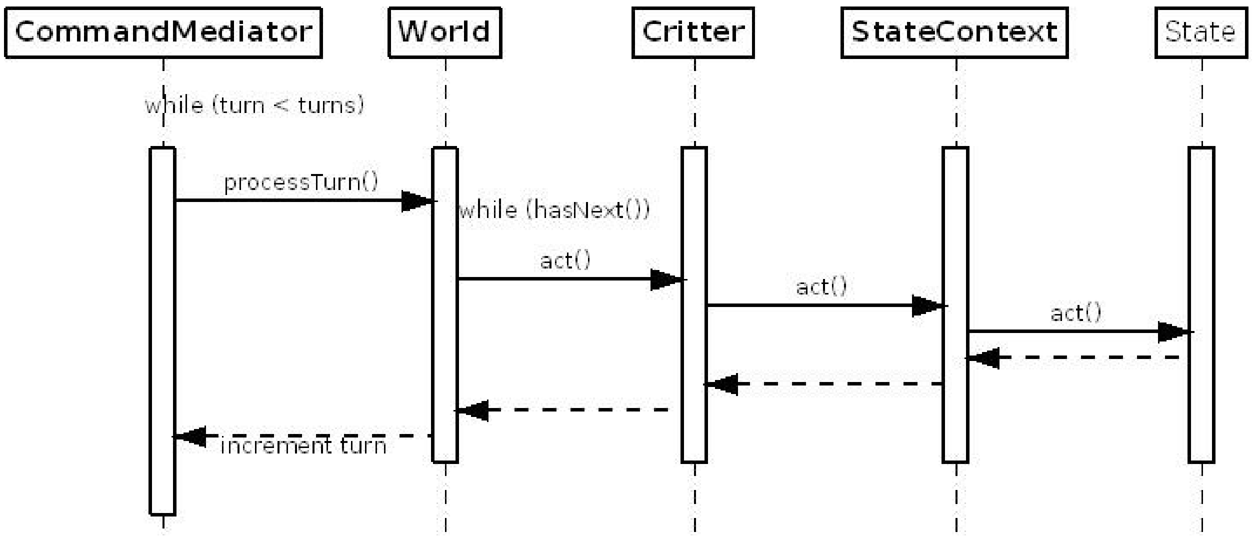
3.4.9 Reproducing State Pseudocode

```
• if PLANT
    ○ if candidate mate in the world found
        ▪ sexual reproduction
    ○ else
        ▪ asexual reproduction (clone)
• else
    ○ if candidate mate in the world found
        ▪ sexual reproduction
    ○ else
        ▪ Change State to Moving
```

3.4.10 Moving Pseudocode

```
• if candidate empty spots found in range
    ○ move to random candidate spot
• else
    ○ do not move
```

3.4.11World Sequence Diagram



Note: Critters are processed through the World grid in serial starting from (0,0) and proceeding row by row until the entire grid has been traversed.

3.5 Utility Classes

The `org.snowcrash.utility` package encompasses classes that are not directly related to any other classes but are used by multiple classes for their utility and functionality.

3.5.1Package Summary

- `org.snowcrash.utility`

Callback	Interface for passing logic to another method.
CloningUtility	Utility that clones a given object by violating Java restrictions.
Constants	Universal constants used by multiple classes.
Observable	Interface that exposes the methods of Java's Observable class.
Pair	Models a pair of generic data.
RandomNumbers	Provides a common means to generate random numbers or Pairs of random numbers.

3.6 Main Classes

The `Main` package encompasses classes that launch, run, and test the application.

3.6.1Package Summary

- No package

Main	The main thread of execution for the application. Initializes global references.
Test	Maintains unit test cases for ease of testing functionality.

4. User Interface Design

This section describes the User Interface, describing and showing concrete versions of the screens and how they interact with each other.

4.1 Description of the User Interface

There are three screens in the UI: the configuration screen which is the first screen that appears when the user starts the application; the simulation screen which runs the simulation; and the results screen which displays the results of a simulation. Each screen consists of the window frame, menu bar, and central panel.

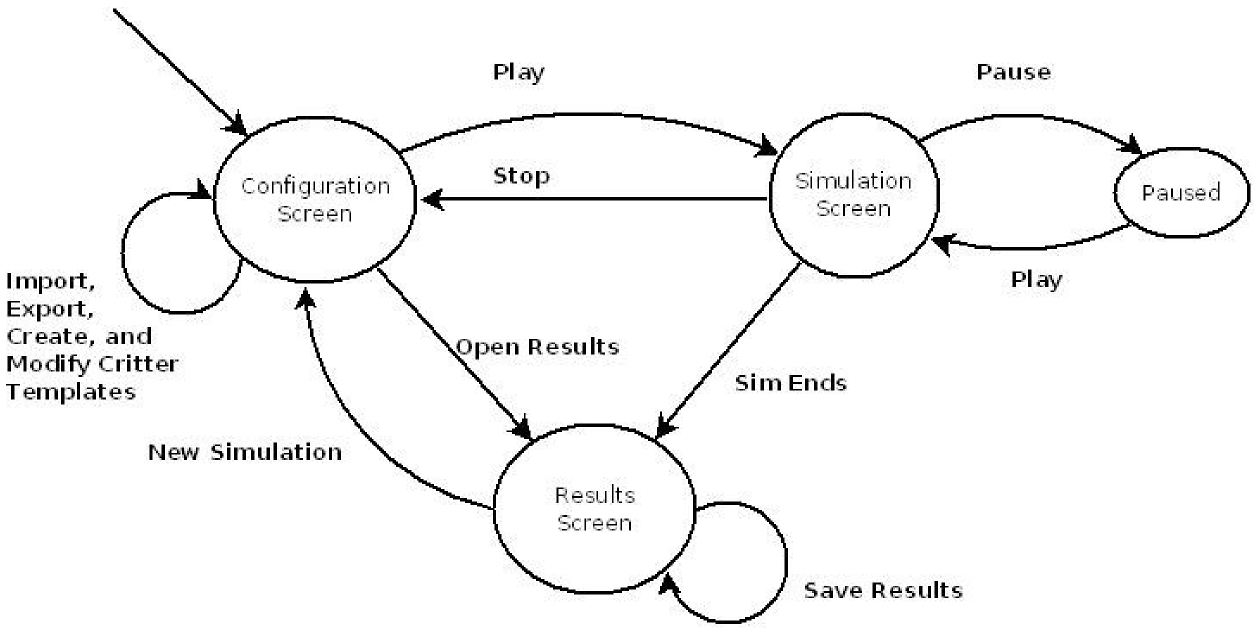
From the configuration screen, the user can either start or load a saved simulation, and go to the simulation screen, or the user can load saved results and go to the results screen.

From the simulation screen, the user can return to the configuration screen to define a new simulation or stop the simulation and view the results.

From the results screen, the user can return to the configuration screen.

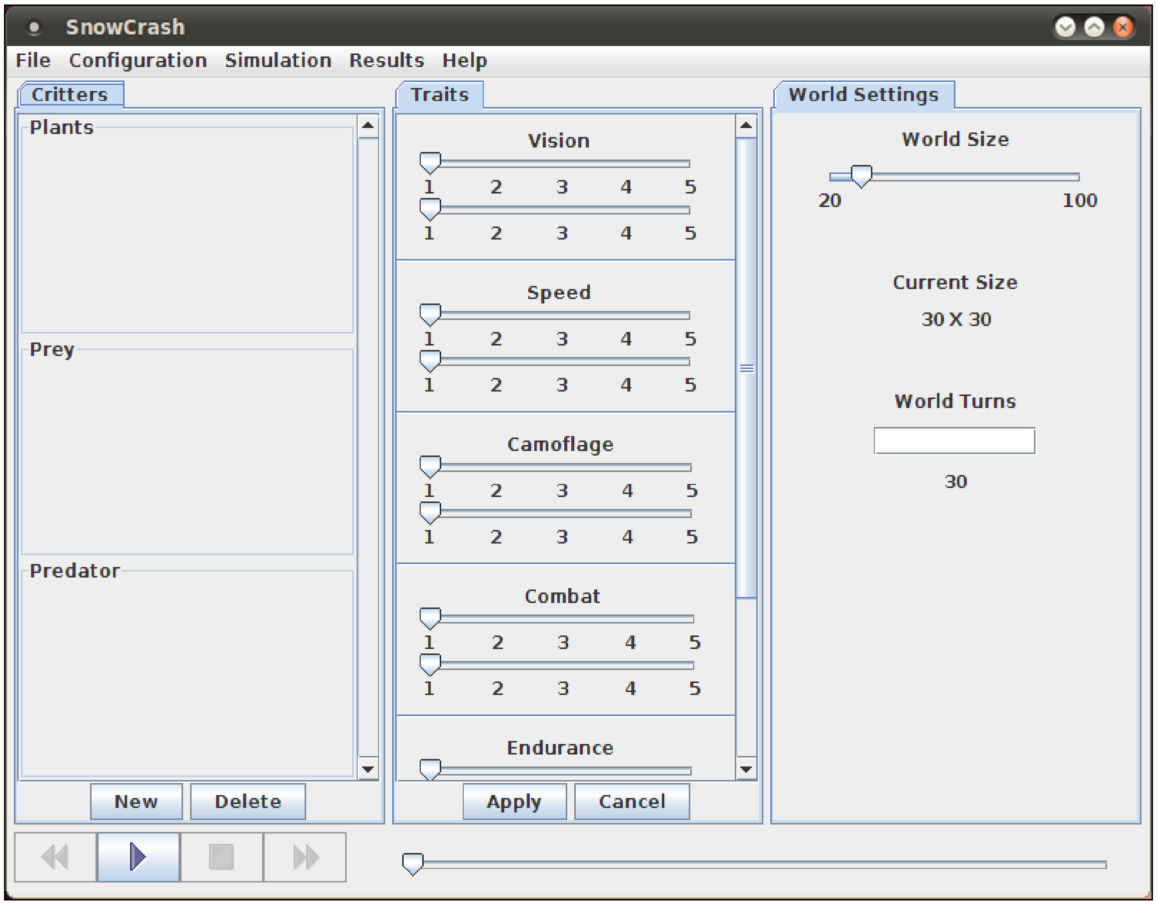
Note: Please refer to the SRS for complete details of the UI functionality. The results screen is not shown among the screen shots as it has not been prototyped yet.

4.2 Screen Transition State Diagram



4.3 Screen Images

4.3.1 Configuration Screen



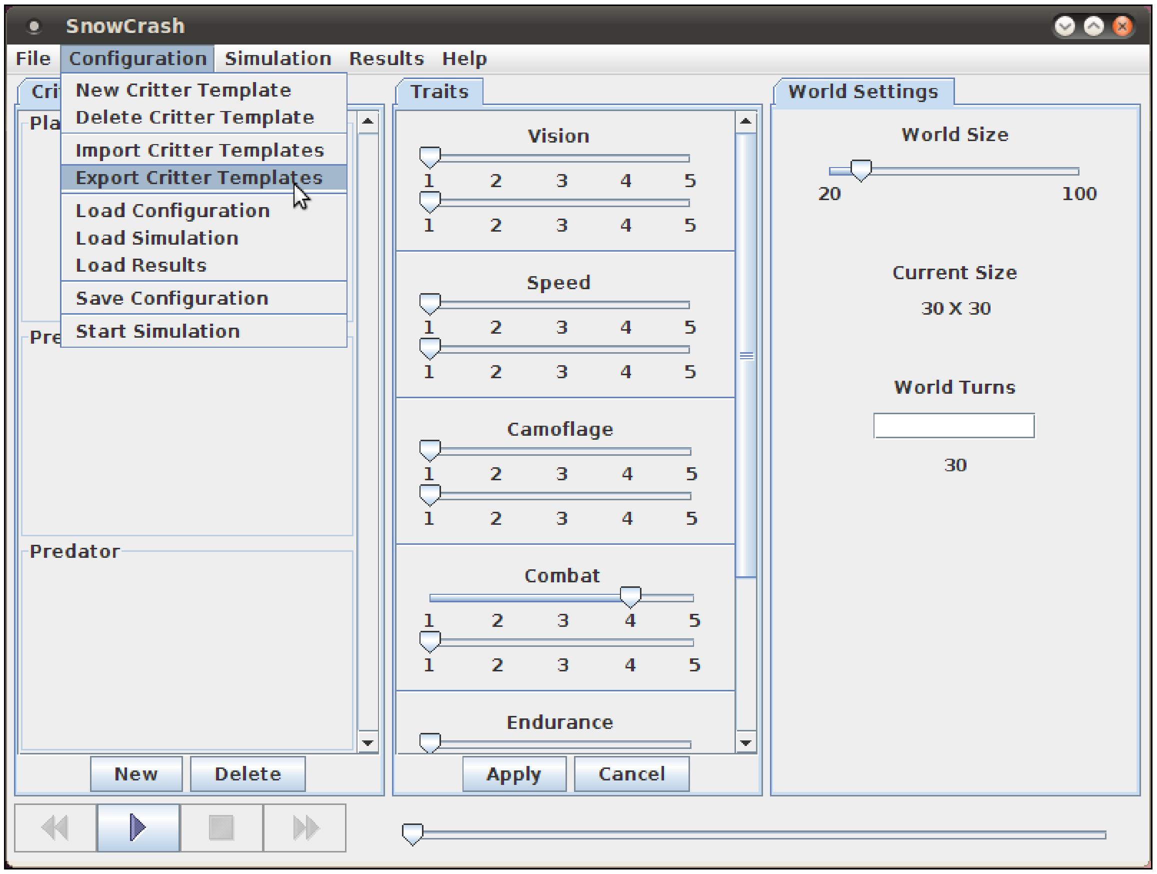
Current Size

30 X 30

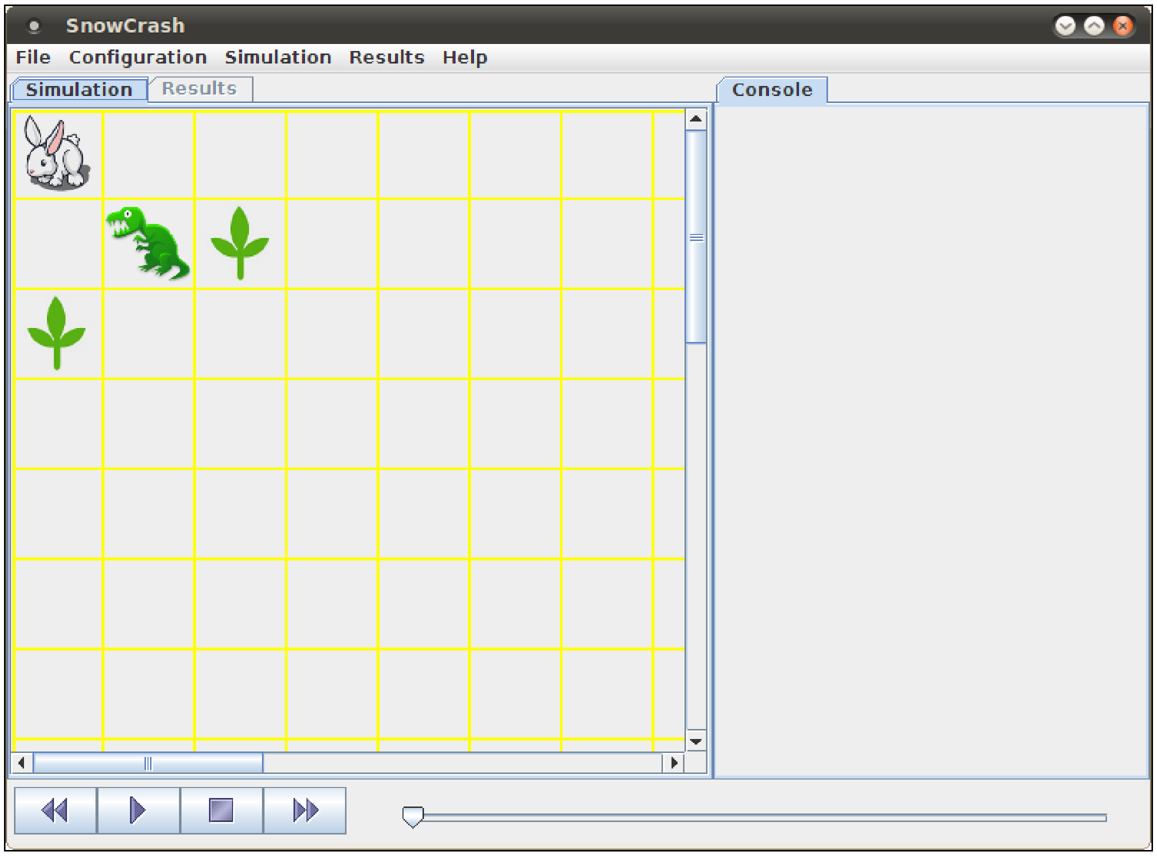
World Turns

30

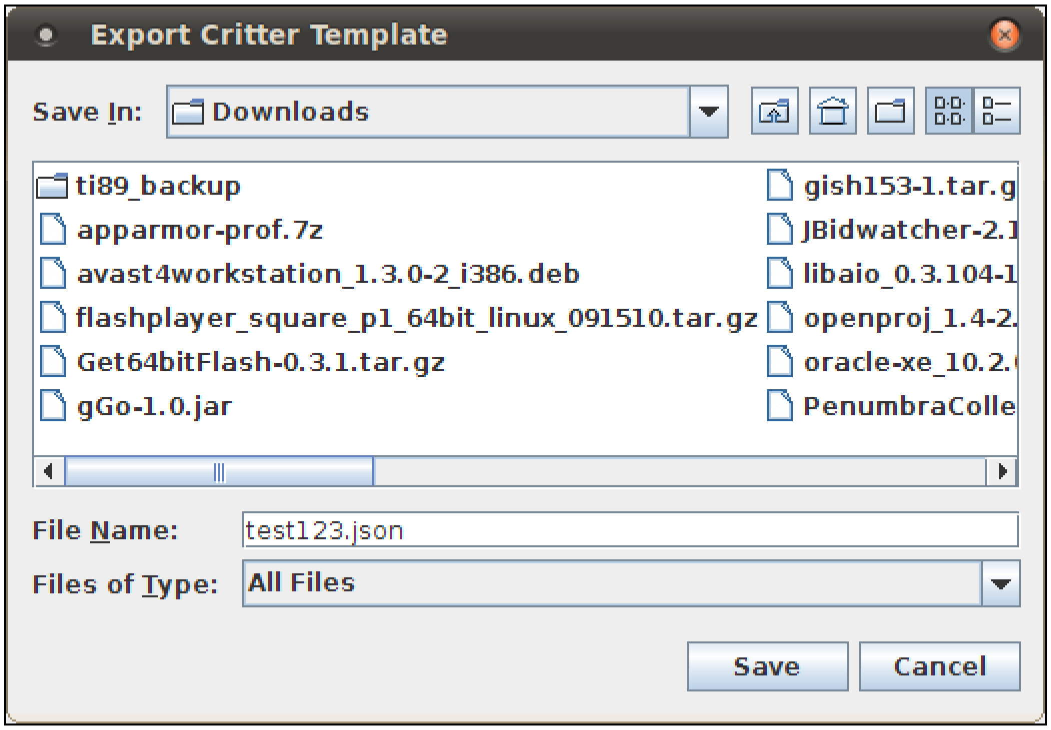
4.3.2 Configuration Screen with File Menu Items



4.3.3 Simulation Screen



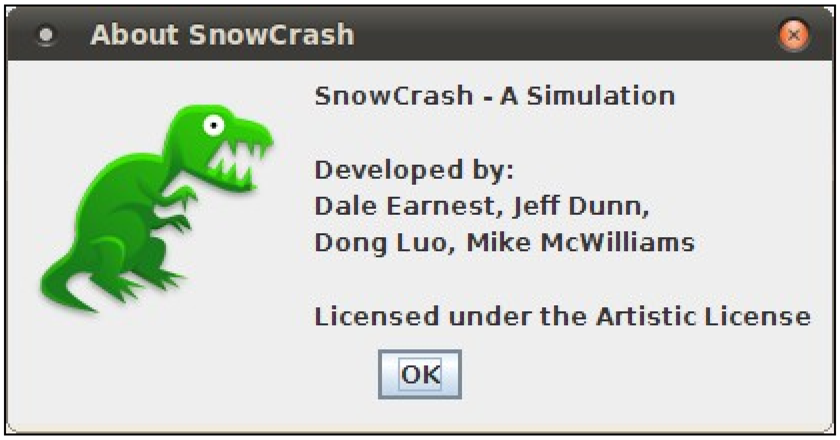
4.3.4Critter Template Export To File Dialog Box



File Name: test123.json

Files of Type: All Files

4.3.5 About Snow Crash Dialog Box

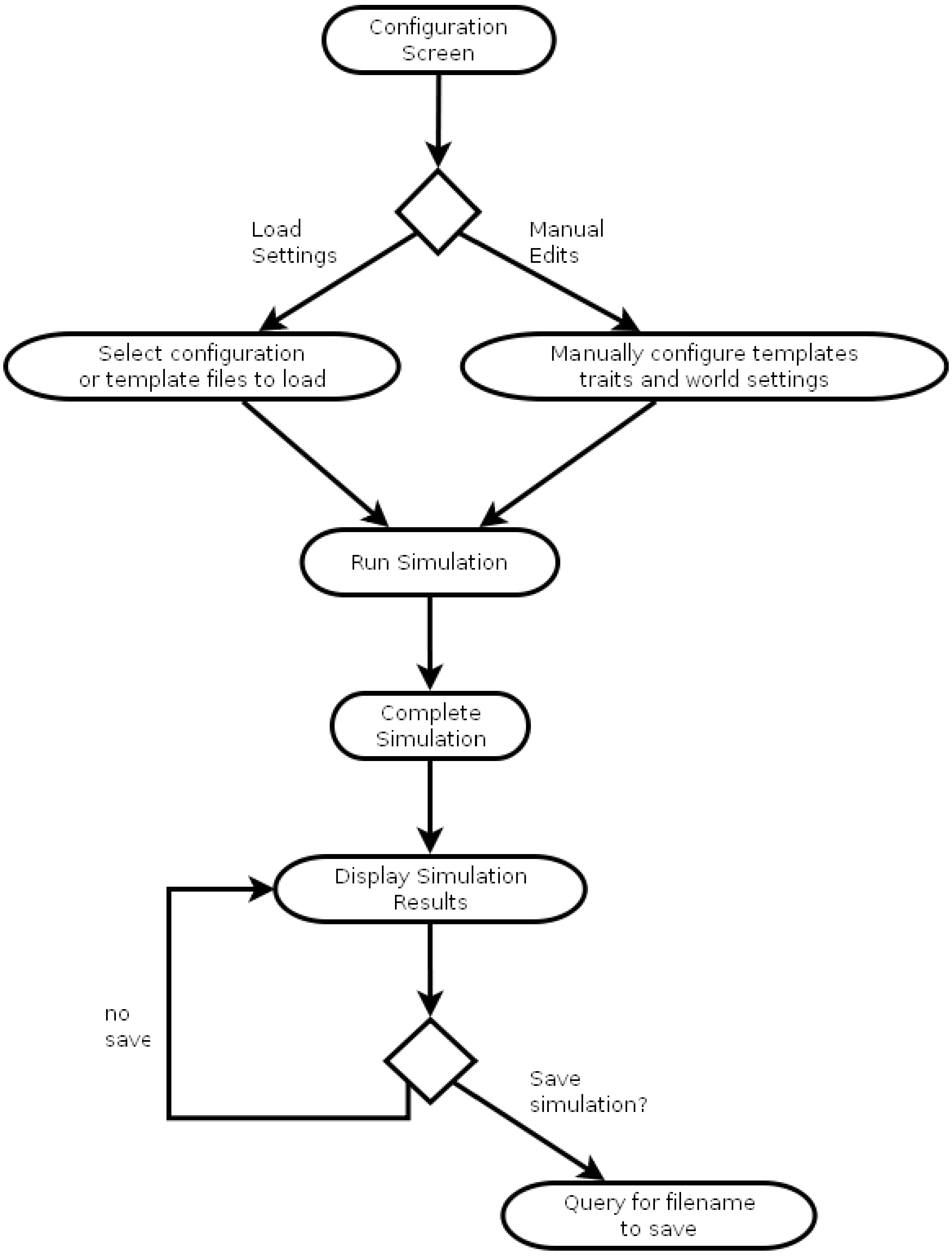


SnowCrash - A Simulation

Developed by:
Dale Earnest, Jeff Dunn,
Dong Luo, Mike McWilliams

Licensed under the Artistic License

4.4 UI Activity Diagram


Configuration Screen
Load Settings
Manual Edits
Select configuration or template files to load
Manually configure templates traits and world settings
Run Simulation
Complete Simulation
Display Simulation Results
no save
Save simulation?
Query for filename to save

5. Post Mortem

5.1 Deliverable Effort

Deliverable	Estimated Hours	Actual Hours
STCD	30	25
SPD	40	38
SCMP	40	57
SPMP	40	62
SRS	60	62
SDD	40	103
Group Project Presentation	30	
Individual Project Report	10	
Totals	290	347

5.2 Individual Contribution

Item	Dale Earnest	Mike McWilliams	Jeff Dunn	Dong Luo	Total Hrs.
Meetings	6	6	6	6	24
Communications Plan	0	0	0	0	0
Application Investigation	3	6	2	6	17
Prototype (coding)	4	14	8	8	34
Prototype (drawings)	0	0	0	0	0
Project Proposal	0	0	0	0	0
Document Editing	10	8	2	3	23
Document Review	1	2	1	1	5
Total Hours	24	36	19	24	103

5.3 Team Effectiveness

5.3.1Team Meeting

Score: 10

Details: Meetings are effective, on topic, with little distraction. Team members are focused, contributing, and eager to complete tasks on the agenda.

5.3.2Document Preparation

Score: 5

Details: Communication is good but timeliness needs improvement.   Real life time factors, other classes, and a drive to hit code milestones have affected the quality of the documentation.

5.3.3Development

Score: 7

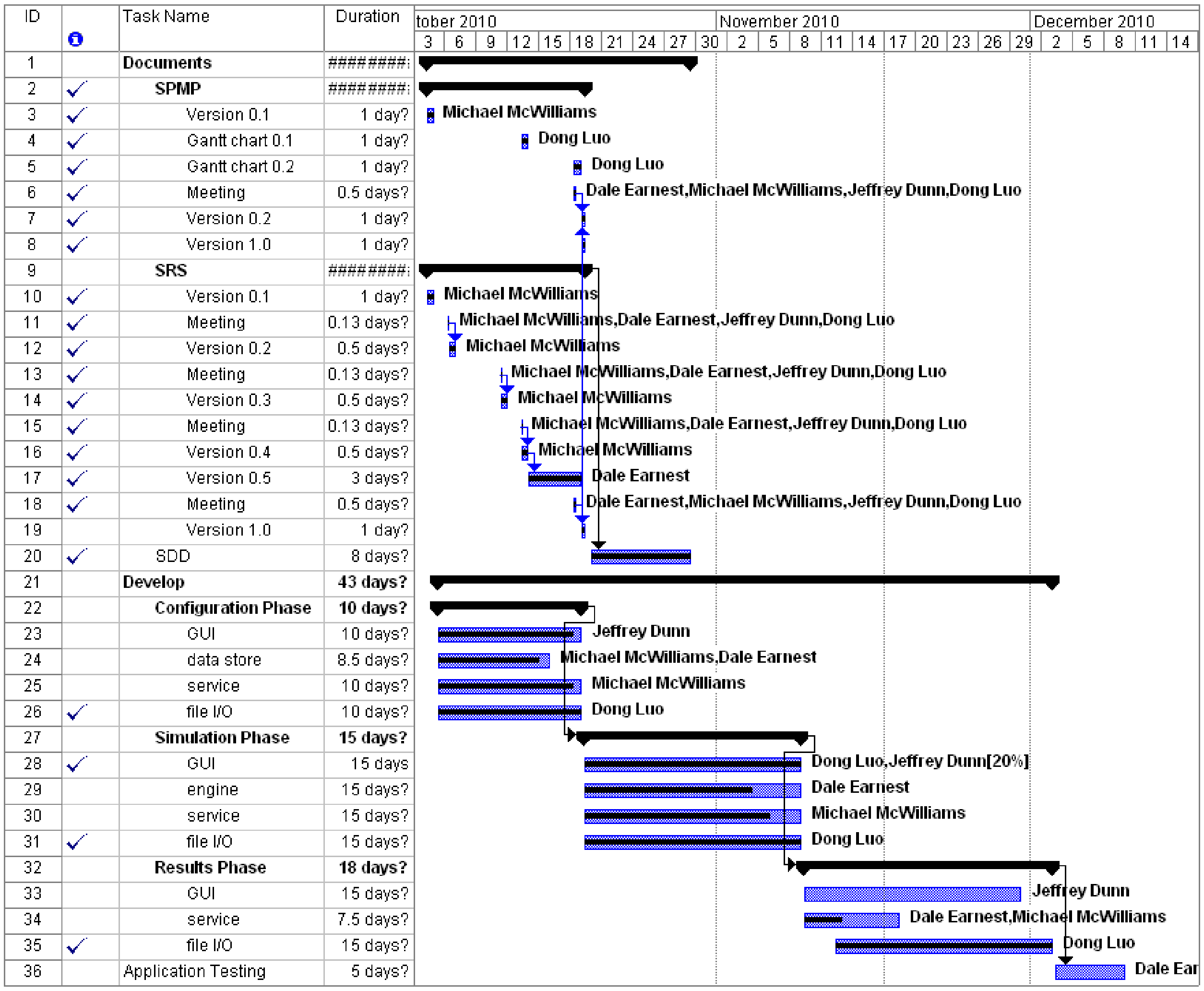
Details: Still behind schedule but we're catching up. Code quality is good and communication between team members is still high, but real life time factors.

5.3.4Summary

Score: 22 out of 30

Details: This score indicates that the project is in a warning zone, that though we have some strong team members our documentation is suffering as we've made a development push.

5.4 Gantt Chart



ID	ℹ	Task Name	Duration
1		Documents	########:
2	✓	SPMP	########:
3	✓	Version 0.1	1 day?
4	✓	Gantt chart 0.1	1 day?
5	✓	Gantt chart 0.2	1 day?
6	✓	Meeting	0.5 days?
7	✓	Version 0.2	1 day?
8	✓	Version 1.0	1 day?
9		SRS	########:
10	✓	Version 0.1	1 day?
11	✓	Meeting	0.13 days?
12	✓	Version 0.2	0.5 days?
13	✓	Meeting	0.13 days?
14	✓	Version 0.3	0.5 days?
15	✓	Meeting	0.13 days?
16	✓	Version 0.4	0.5 days?
17	✓	Version 0.5	3 days?
18	✓	Meeting	0.5 days?
19		Version 1.0	1 day?
20	✓	SDD	8 days?
21		Develop	43 days?
22		Configuration Phase	10 days?
23		GUI	10 days?
24		data store	8.5 days?
25		service	10 days?
26	✓	file I/O	10 days?
27		Simulation Phase	15 days?
28	✓	GUI	15 days
29		engine	15 days?
30		service	15 days?
31	✓	file I/O	15 days?
32		Results Phase	18 days?
33		GUI	15 days?
34		service	7.5 days?
35	✓	file I/O	15 days?
36		Application Testing	5 days?

tober 2010
November 2010
December 2010
3 6 9 12 15 18 21 24 27 30 2 5 8 11 14 17 20 23 26 29 2 5 8 11 14
Michael McWilliams
Dong Luo
Dong Luo
Dale Earnest,Michael McWilliams,Jeffrey Dunn,Dong Luo
Michael McWilliams
Michael McWilliams,Dale Earnest,Jeffrey Dunn,Dong Luo
Michael McWilliams
Michael McWilliams,Dale Earnest,Jeffrey Dunn,Dong Luo
Michael McWilliams
Michael McWilliams,Dale Earnest,Jeffrey Dunn,Dong Luo
Michael McWilliams
Dale Earnest
Dale Earnest,Michael McWilliams,Jeffrey Dunn,Dong Luo
Jeffrey Dunn
Michael McWilliams,Dale Earnest
Michael McWilliams
Dong Luo
Dong Luo,Jeffrey Dunn[20%]
Dale Earnest
Michael McWilliams
Dong Luo
Jeffrey Dunn
Dale Earnest,Michael McWilliams
Dong Luo
Dale Ear

Milestones Accomplished

- SDD is complete.
- Team Meetings have met on time.
- PR 0.1 is 90% complete.
- Simulation Screen UI duty was reassigned to Dong and is complete.
- Critter States is 80% complete and almost ready for integration testing.