Design Description for Project 3: Heated Planet

by

Team 1

Michael Ashby - (mashby6), Robert Benfield - (rbenfield7), Ross Broderson - (rbroderson3), Aaron Higdon - (ahigdon3), Anthony Lozano - (alozano3)

Georgia Institute of Technology

CS 6310 Software Architecture and Design
Project 3
December 1, 2014

Overview

In this project we are expanding on the heated plate and heated earth project by adding new functional and nonfunctional requirements. The functional requirements were expanded to include a tilted axis and an elliptical orbit around a heat source, it's Sun. Non functional requirements include persistence of data as the simulation runs and the ability to retrieve and interpolate from that data when called to do so, greater transparency which will hide the source of result data from the user, improved user experience through an enhanced user interface and architecture best practices including design patterns. The purpose of this project is to analyze the impact of additional functional and nonfunctional requirements such as orbit, tilt, persistence and transparency has on software architecture design, performance and results.

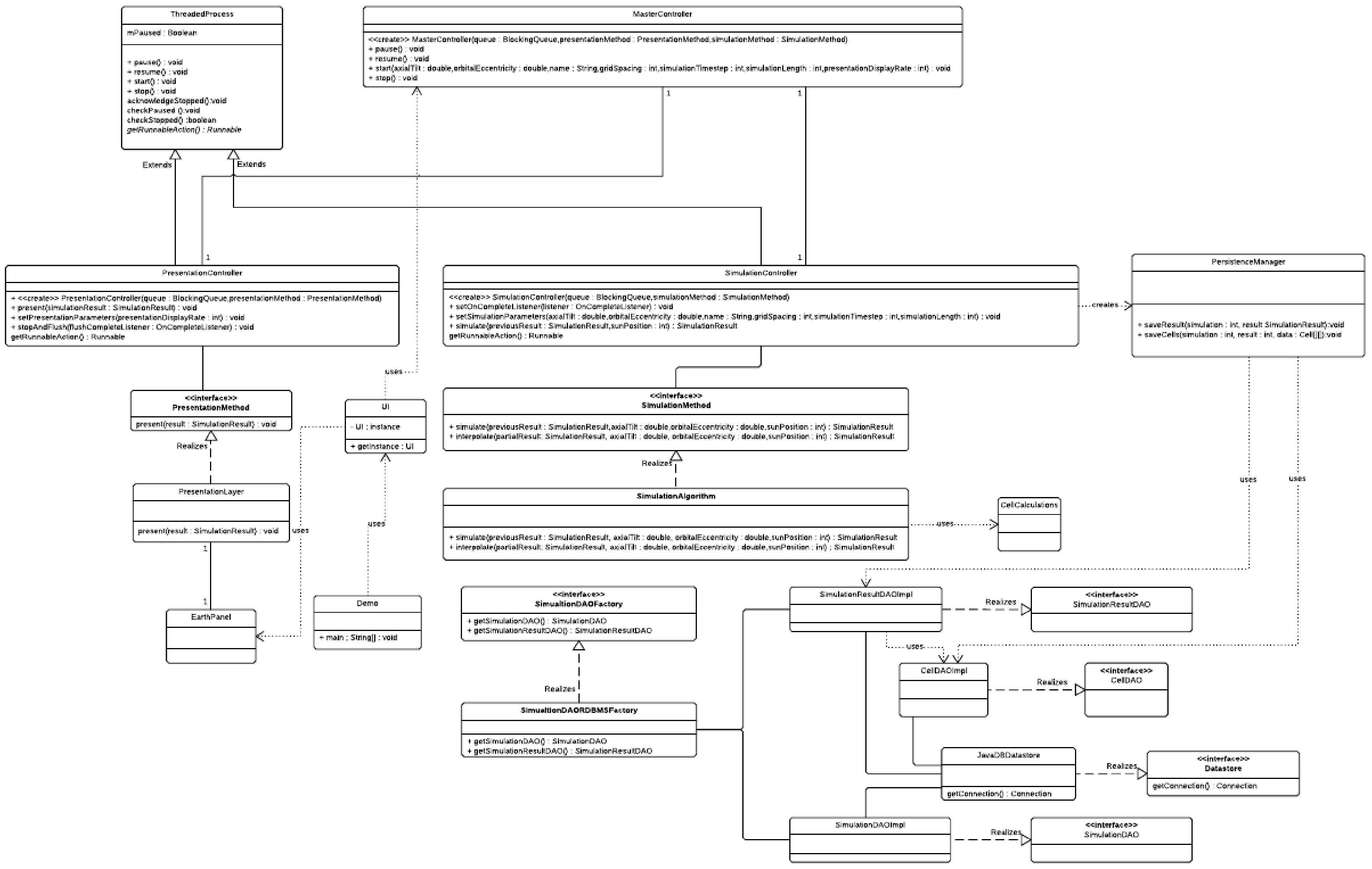
The program is written in Java using object oriented techniques. One of the major goals of the architecture is to provide the ability for users to query the system for existing simulations that match their criteria. If found, the system should display the simulation to the user in the same manner as a new simulation and interpolate any missing data but remain transparent to the user. The presentation of the simulation must show the temperature fluctuations as the planet rotates around the heat source. In order to meet these requirements, the model-view-controller pattern was chosen in order to decouple the UI from the representation increasing flexibility and reuse which enables us to hide the implementation from the user and meet the requirements for transparency.

The controller classes for the simulation and presentation described in greater detail below operate in their own individual threads and are managed by a master controller. Additionally, the program architecture leverages design patterns such as the FactoryMethod and AbstractFactory to allow support for mock objects used for testing purposes and the ability to support multiple different data persistence methods such as databases, flat files or other.

UML Class Model design diagram

The classes used in the PlanetSim simulation program were laid out in 3 particular design element types: View components, Logic components and Persistence components. The view components were responsible for user interaction and graphical representation of the simulation. The Logic components were responsible for organization of the overall program execution and computational steps required in performing the simulation. The Persistence components were responsible for storing the simulation data and producing the stored results on command.

Figure 1: UML Class Model design diagram (following page)


ThreadedProcess
mPaused : Boolean
+ pause() : void
+ resume() : void
+ start() : void
+ stop() : void
acknowledgeStopped():void
checkPaused ():void
checkStopped() :boolean
getRunnableAction() : Runnable
MasterController
<<create>> MasterController(queue : BlockingQueue,presentationMethod : PresentationMethod,simulationMethod : SimulationMethod)
+ pause() : void
+ resume() : void
+ start(axialTilt : double,orbitalEccentricity : double,name : String,gridSpacing : int,simulationTimestep : int,simulationLength : int,presentationDisplayRate : int) : void
+ stop() : void
1
1
Extends
Extends
1
1
PresentationController
+ <<create>> PresentationController(queue : BlockingQueue,presentationMethod : PresentationMethod)
+ present(simulationResult : SimulationResult) : void
+ setPresentationParameters(presentationDisplayRate : int) : void
+ stopAndFlush(flushCompleteListener : OnCompleteListener) : void
getRunnableAction() : Runnable
SimulationController
<<create>> SimulationController(queue : BlockingQueue,simulationMethod : SimulationMethod)
+ setOnCompleteListener(listener : OnCompleteListener) : void
+ setSimulationParameters(axialTilt : double,orbitalEccentricity : double,name : String,gridSpacing : int,simulationTimestep : int,simulationLength : int) : void
+ simulate(previousResult : SimulationResult,sunPosition : int) : SimulationResult
getRunnableAction() : Runnable
creates
PersistenceManager
+ saveResult(simulation : int, result SimulationResult):void
+ saveCells(simulation : int, result : int, data : Cell[][]):void
uses
uses
<<interface>>
PresentationMethod
present(result : SimulationResult) : void
Realizes
PresentationLayer
present(result : SimulationResult) : void
1
1
EarthPanel
uses
UI
- UI : instance
+ getInstance : UI
uses
Demo
+ main ; String[] : void
uses
<<interface>>
SimulationMethod
+ simulate(previousResult : SimulationResult,axialTilt : double,orbitalEccentricity : double,sunPosition : int) : SimulationResult
+ interpolate(partialResult: SimulationResult, axialTilt : double, orbitalEccentricity : double,sunPosition : int) : SimulationResult
Realizes
SimulationAlgorithm
+ simulate(previousResult : SimulationResult, axialTilt : double, orbitalEccentricity : double,sunPosition : int) : SimulationResult
+ interpolate(partialResult: SimulationResult, axialTilt : double, orbitalEccentricity : double,sunPosition : int) : SimulationResult
uses
CellCalculations
<<interface>>
SimualtionDAOFactory
+ getSimulationDAO() : SimulationDAO
+ getSimulationResultDAO() : SimulationResultDAO
Realizes
SimualtionDAORDBMSFactory
+ getSimulationDAO() : SimulationDAO
+ getSimulationResultDAO() : SimulationResultDAO
SimulationResultDAOImpl
Realizes
<<interface>>
SimulationResultDAO
uses
CellDAOImpl
Realizes
<<interface>>
CellDAO
JavaDBDatastore
getConnection() : Connection
Realizes
<<interface>>
Datastore
getConnection() : Connection
SimulationDAOImpl
Realizes
<<interface>>
SimulationDAO

Static Description

The Demo class's primary function is to parse the command-line arguments, validate the argument values and invoke the UI Class. The Demo class creates an instance of the MasterController class by calling the getMasterController method of the ObjectFactory class. The UI class is initialized with the instance of the MasterController.

Dependency Type	**UML Stereotype**	**Description**
Instantiates	«instantiate»	UI
Calls	«call»	Utils.parseArguments
uses as parameter		args: String[]
Creates	«create»	ObjectFactory
Instantiates	«instantiate»	MasterController

Figure 2: PlanetSim.Demo Dependency Table

The ObjectFactory creates new instances of PresentationLayer and SimulationMethod when instantiated by the UI class and defines an ArrayBlockingQueue with a maximum capacity of 10 for storing classes of type SimulationResult.

Dependency Type	UML Stereotype	Description
Instantiates	«instantiate»	BlockingQueue
Instantiates	«instantiate»	PresentationMethod
Instantiates	«instantiate»	SimulationMethod
Instantiates	«instantiate»	SimulationResult

Instantiates	«instantiate»	SimulationDAO
Instantiates	«instantiate»	CellDAO
Instantiates	«instantiate»	SimulationResultDAO
Instantiates	«instantiate»	SimulationDAOFactory

Figure 3:base.ObjectFactory Dependency Table

The UI class is responsible for generating the user interface which provides a set of controls for collecting user input and buttons for controlling the simulation. Action listeners on the click of start, stop, pause, resume and simulation complete are managed within the MasterController class which instantiates the SimulationController Class and the PresentationController if user desired.

Dependency Type	UML Stereotype	Description
Instantiates	«instantiate»	PresentationController
Instantiates	«instantiate»	SimulationController
uses as parameter		BlockingQueue<SimulationResult> queue, PresentationMethod presentationMethod, SimulationMethod simulationMethod

Figure 4: controllers.MasterController Dependency Table

The PresentationController class receives a SimulationResult and the PresentationLayer class and manages the display.

Dependency Type	UML Stereotype	Description
uses as parameter		BlockingQueue<SimulationResult> queue, PresentationMethod presentationMethod (PresentationLayer)

Figure 5: controllers.PresentationController Dependency Table

The SimulationController class receives a SimulationResult and SimulationMethod which will be of type MockSimulationMethod or SimulationAlgorithm. The controller manages the simulation thread and determines if the simulation is new or exists. If the simulation type indicator is new, the controller will create a new instance of the Simulation class otherwise the existing simulation is used. New simulations are saved to the database by using the saveResult method of the PersistanceManager class. The SimulationMethod interface defines two different methods that return a SimulationResult. The simulate method receives a previous result and input parameters in order to conduct a new simulation. The interpolate method receives a previous result and a partial result so that it can fill in any missing values which will result in a complete simulation when viewed by the user in the presentation.

Dependency Type	UML Stereotype	Description
Instantiates	«instantiate»	Simulation
Instantiates	«instantiate»	PersistanceManager

uses as parameter		BlockingQueue<SimulationResult> queue, SimulationMethod simulationMethod
uses as parameter		axialTilt, orbitalEccentricity, name, gridSpacing, simulationTimestep, simulationLength

Figure 6: controllers.SimulationController Dependency Table

Both of the controller classes for simulation and presentation extends the ThreadedProcess in order to manage each process within its own thread. The SimulationAlgorithm class which implements the SimulationMethod interface is responsible for conducting the simulation calculations and returning the results in a SimulationResult class.

Dependency Type	UML Stereotype	Description
Instantiates	«instantiate»	CellCalculations
Instantiates	«instantiate»	OrbitalPosition
uses as parameter		SimulationResult previousResult, double axialTilt, double orbitalEccentricity, int sunPosition, int gridSpacing, double planetCircumference, double planetAldebo, double

		planetEmissivity, double orbitSemiMajorAxis, double solarYear, double solarPowerPerMeter
returns as result		SimulationResult

Figure 7: simulation.SimulationAlgorithm Dependency Table

There are two tabs that allow the user to start a new simulation or query the database for an existing simulation. Either case will create an instance of the EarthPanel class which presents the results to the user. If the user chooses to load an existing simulation, the ObjectFactory class is used to create the SimulationDAOFactory class which will return the correct type of subclass depending on the type specified which can be RDBMS (prior simulations stored in the database), MOCK (hard coded classes used for testing) or FLATFILE (not currently implemented). Interfaces are defined for the simulation, simulation result and cell classes so that the sub classes for the different types properly provide the necessary methods for persisting result data.

Dependency Type	UML Stereotype	Description
Instantiates	«instantiate»	EarthPanel
Creates	«create»	ObjectFactory
Creates	«create»	Simulation

Figure 8:gui.UI Dependency Table

Dependency Type	UML Stereotype	Description
Instantiates	«instantiate»	SimulationRDBMSDAOFactoryImpl
Creates	«instantiate»	MockSimulationDAOFactory

Figure 9:data.SimulationDAOFactory Dependency Table

The SimulationRDBMSDAOFactoryImpl class and MockSimulationDAOFatory class are concrete implementations of the abstract factory SimulationDAOFactory class, responsible for initializing classes needed for retrieving and storing (in the case of RDBMS) data for the simulation, simulation results and cell data.

Dependency Type	UML Stereotype	Description
Instantiates	«instantiate»	SimulationRDBMSDAO
Instantiates	«instantiate»	SimulationResultRDBMSDAO
Instantiates	«instantiate»	CellRDBMSDAO

Figure 10:data.impl.SimulationRDBMSDAOFactoryImpl Dependency Table

Dependency Type	UML Stereotype	Description
Instantiates	«instantiate»	MockSimulationDAO
Instantiates	«instantiate»	MockSimulationResultDAO

Figure 11:data.impl.MockSimulationDAOFactory Dependency Table

Dependency Type	UML Stereotype	Description
Instantiates	«instantiate»	EarthGridDisplay

Instantiates	«instantiate»	SunDisplay
Instantiates	«instantiate»	TimeDisplay
uses as parameter		Dimension: minSize, maxSize, prefSize

Figure 12:gui.EarthPanel Dependency Table

Design Rationale

The simulation result data was stored in an embedded Apache Derby Database. The database was selected because it met the requirements for how the data needed to be queried, stored and retrieved. Additionally, the database selected was easy to use, free of cost and easily met the storage space requirements for the program. Since the database is accessed by only one instance of the program, the concern for database concurrency was entirely contained to the programs active threads. To manage concurrent database operations, thread local storage was utilized to manage a single connection to the database per operating thread.

Data was stored as individual attributes which helped with the query language and simplified parsing of data on save and load operations. The data model was normalized and enforced referential integrity with the assignment of primary and foreign key relationships.

Concurrency was managed in the program by implementing a master controller class that instantiated a simulation and presentation controller class. Each controller operated within it's own thread. As data results were generated during a simulation, the simulation controller would add the results to a queue (BlockingQueue) that acts on a first in first out (FIFO) basis. The presentation controller requests a new simulation result when ready. The queue mechanism prevents any contention issues when adding or loading results.

Saved simulations were available to be searched using simulation name and/or physical factors of the simulation. Additionally, the results were capable of being scoped by time period and geographical location. In the event multiple simulations matched a user's search criteria, results were returned to the query interface ordered by the smallest grid size, precision, temporal precision and geographical precision. The ordered list allows the user interface to choose the best fit, in the case of this implementation the first result is chosen.

Medium	Ease of setup	Relational Model Support	Queryable	Data Storage	Cost
Apache Derby Database	Good	Excellent	Excellent	Excellent	Free
Flat File	Good	Poor	Poor	Excellent	Free

Figure 13: Persistence Medium Selection Criteria

Medium	Ease of use	Memory Use	Queryable
Relational Tables	Good	Excellent	Excellent
XML	Good	Poor	Poor

Figure 14: Data Format Selection Criteria

Logical Data Model

The program provides the ability for the user to query the system for previously simulations that have already been created. In order to meet this need, the logical model needs to have a simulation representation that has attributes describing the simulation so that it can be retrieved such as name, date of execution and configuration parameters. The result of the simulation also needs to be stored since it will be used to generate the presentation in the GUI. The simulation will generate many results within a defined time period as the planet is in orbit around the heat source. The planet is broken down into small areas using the latitude and longitude and are needed to determine the amount of heat gain and loss at given times when it is in a different position relative to the heat source's position. These smaller areas are stored in the cell relation which has a many to one relationship with Result and Simulation.

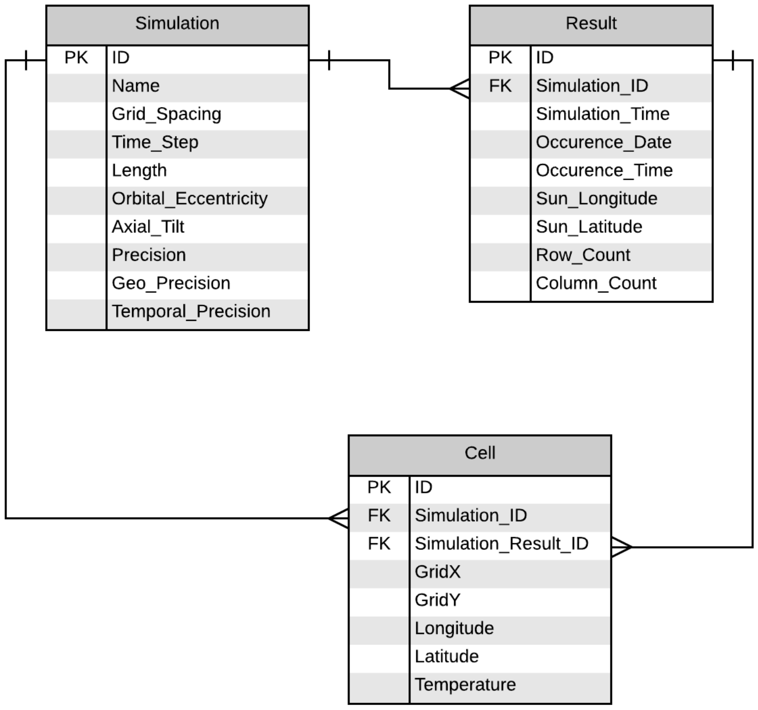


Figure 15: Logical Data Model

Interaction diagrams

As the user operates the program and starts the process by clicking the [Start] button, an instance of the MasterController class instantiates both a PresentationController and SimulationController that run within their own thread respectively. The SimulationController uses the FactoryMethod pattern to determine if the simulation should invoke concrete classses that implement the SimulationMethod interface; the mock method used for testing or if actual calculations should occur using the SimulatiomAlgorithm class. Results are returned to the SimulationController in an instance of the SimulationResult class and in the case when it is a new calculated simulation are saved in a database using the PersistanceManager class.

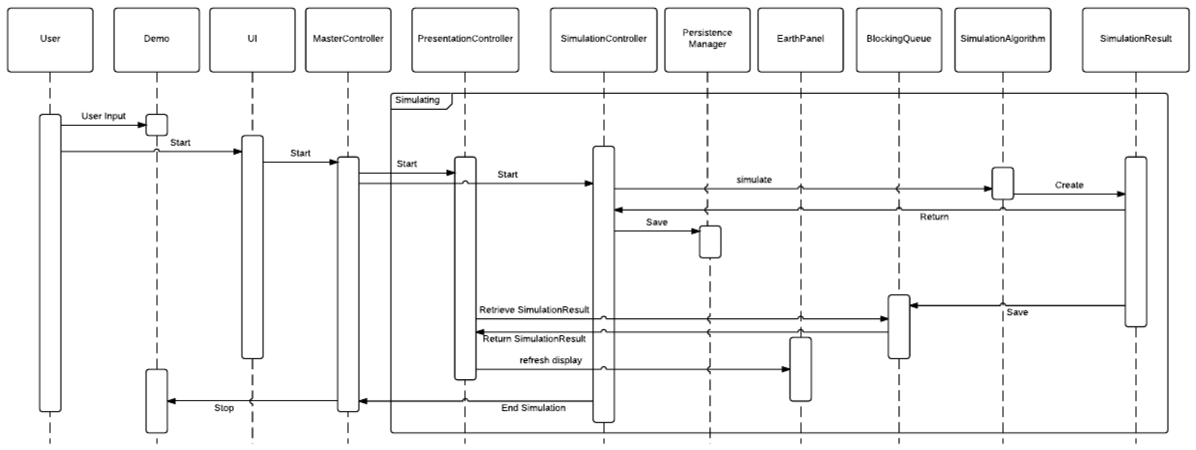


Figure 16: Sequence Diagram for Simulation Process with New Simulation

The user interface also provides the ability for the user to query the database for existing simulations that match the same configuration parameters. In this case, the saved simulation is retrieved from the database and used within the SimulationAlgorithm class to interpolate any missing calculations. The final results are passed back to the SimulationController the same way a new simulation would have been so that the PresentationController class can render the GUI without any modifications or knowledge of the type of simulation that was conducted.

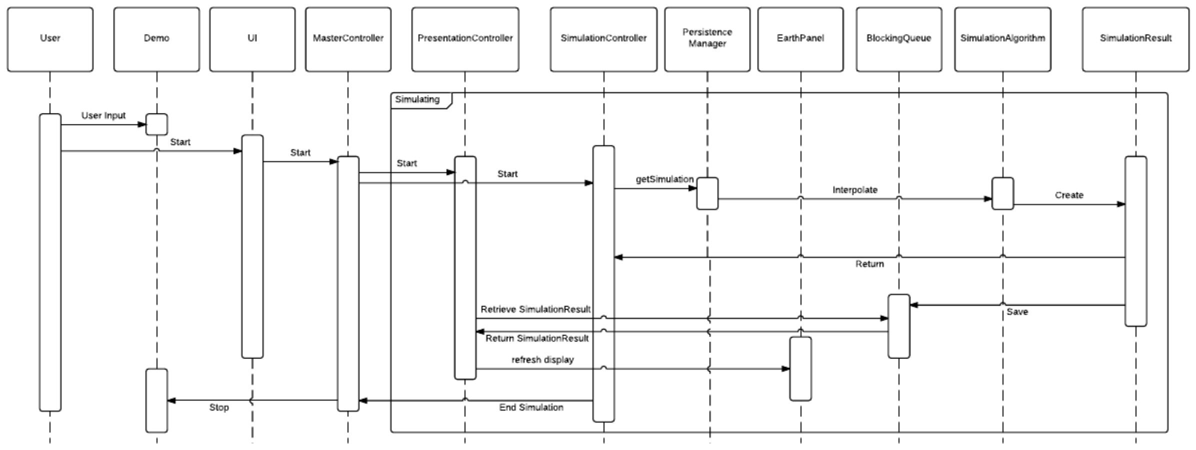


Figure 17: Sequence Diagram for Simulation Process with Interpolation

The following box and arrow diagram shows the high level flow of operations in the program. The Demo class uses a Utils (utilities) class to validate input parameters and invocation arguments to ensure the program runs within expected boundary conditions. The Demo class initializes the UI and MasterController which then responds to event listeners responding to user interactions with the UI such as clicking start, stop and query. The MasterController instantiates both the SimulationController and PresentationController which retrieve and store SimulationResults from the BlockingQueue. The SimulationController saves results including details about the simulation configuration and cell grid values to a data store using the PersistanceManager class. Simulation results are calculated using the SimulationAlgorithm class which understands cell grids, new result sets, previous result sets and uses the CellCalculations class that evaluates cell conditions and returns granular cell results.

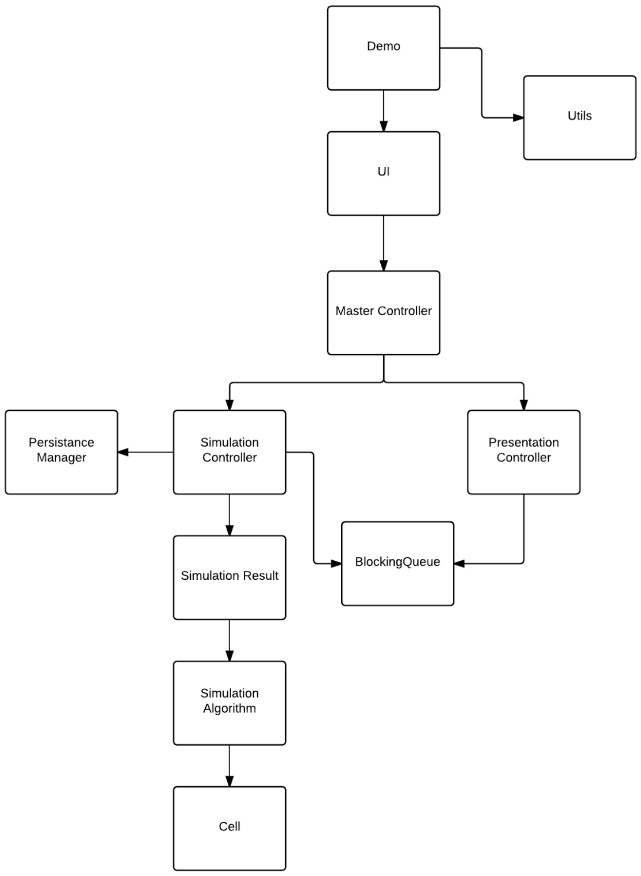


Figure 18: Box and Arrow Diagram of Configuration

The state diagram below models the states and transitions that occur from the click of the start, stop, pause and resume button. A notable difference between the heated planet project and heated earth project is that the user now has the ability to click the [Query] button which will search for existing simulations meeting the criteria. If an existing simulation is found, the program will appear as running to the user and the source of the results being generated transparently.

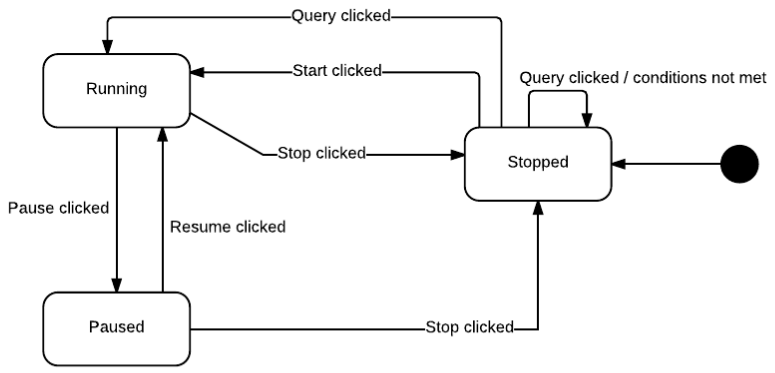
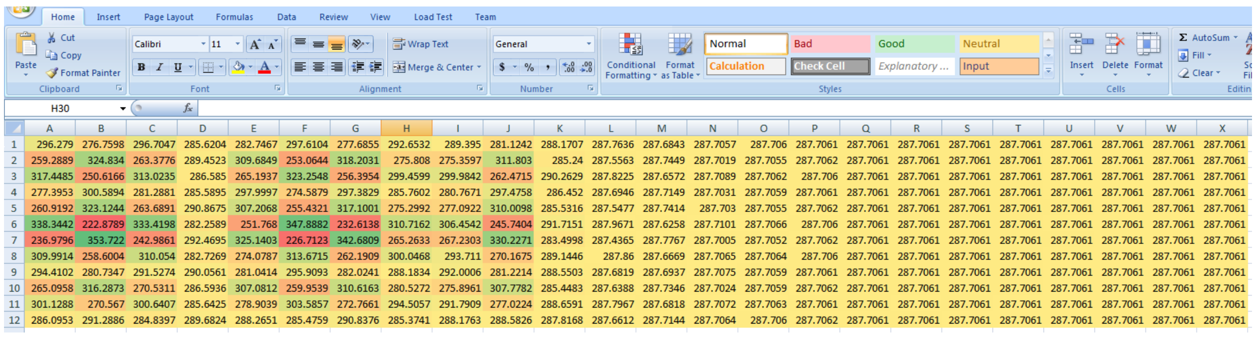


Figure 19: State chart diagram.

GUI Design

The user interface must provide input controls for the user to configure parameters, operate the simulation and receive the results both visually and numerically. The design leveraged previous designs used from teams on the heated earth project where an image of the earth is displayed along with a grid representing areas of latitude / longitude where heating and cooling calculations are performed with respect to the location to the heating source. A future enhancement would be to make the program more generic allow the user to specify the planet and it's heat source where different images and properties would be loaded or made configurable. Since the user has the option of running a new simulation or running a query for existing simulations, the UI includes a tab control that separates the configuration and makes it intuitive for the user to operate either. Regardless of the method of simulation, the visual representation of the results is transparent to the user in that the same image, grid and display of results is conducted. As described previously, the program is managed by a MasterController class which separates the simulation and presentation that run in their own threads and leverage a queue to store and retrieve results. The presentation is managed by the PresentationController class which retrieves simulation results and updates the EarthGrid class that changes the visualization for the user. The team chose this design in order to meet the visualization requirements where the effects of heating and cooling with respect to the location of the heat source is displayed as the simulation runs and displays continuously and smoothly. Input controls include combo boxes that load existing queries, buttons for starting, stopping, pausing and resuming simulations, spinner controls for selecting date & time, check boxes for toggling selections and text fields with validation for taking user input.

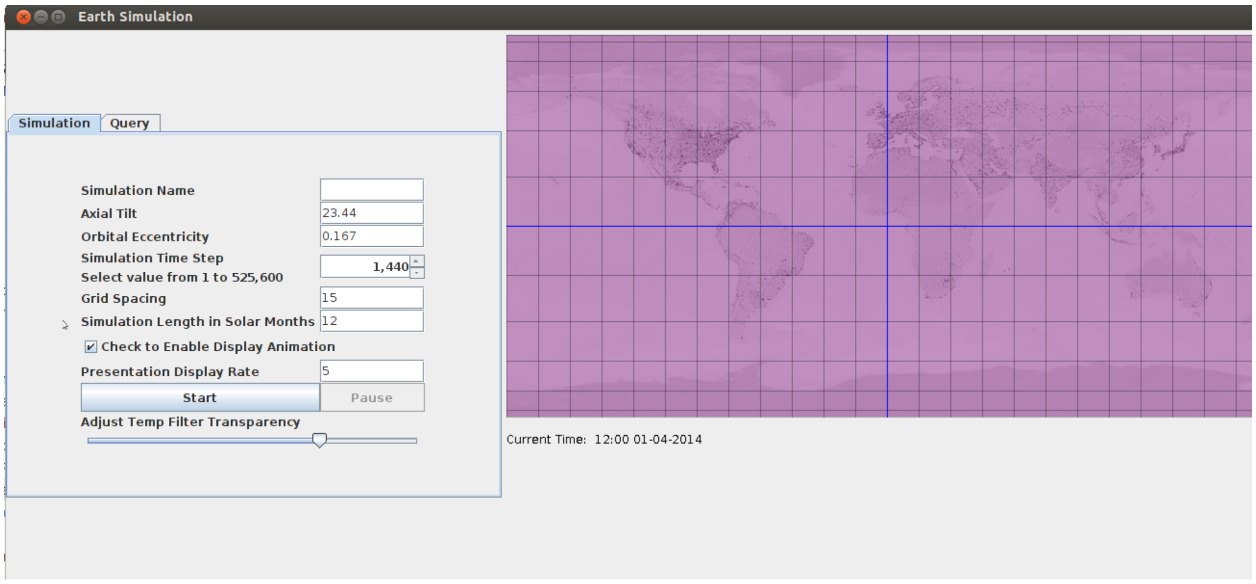
The user can request any combination of views of the results table including minimum temperature, maximum temperature and mean temperature in addition to reproducing the simulation from recorded or interpolated values. In order to meet the ability to display the aggregate results, the user interface includes the ability to download and open files that contain the tabular results. The results can be copied and pasted into a spreadsheet with conditional formatting capabilities to see the results (refer to figure below for example).



	A	B	C	D	E	F	G	H	I	J	K	L
1	296.279	276.7598	296.7047	285.6204	282.7467	297.6104	277.6855	292.6532	289.395	281.1242	288.1707	287.7636
2	259.2889	324.834	263.3776	289.4523	309.6849	253.0644	318.2031	275.808	275.3597	311.803	285.24	287.5563
3	317.4485	250.6166	313.0235	286.585	265.1937	323.2548	256.3954	299.4599	299.9842	262.4715	290.2629	287.8225
4	277.3953	300.5894	281.2881	285.5895	297.9997	274.5879	297.3829	285.7602	280.7671	297.4758	286.452	287.6946
5	260.9192	323.1244	263.6891	290.8675	307.2068	255.4321	317.1001	275.2992	277.0922	310.0098	285.5316	287.5477
6	338.3442	222.8789	333.4198	282.2589	251.768	347.8882	232.6138	310.7162	306.4542	245.7404	291.7151	287.9671
7	236.9796	353.722	242.9861	292.4695	325.1403	226.7123	342.6809	265.2633	267.2303	330.2271	283.4998	287.4365
8	309.9914	258.6004	310.054	282.7269	274.0787	313.6715	262.1909	300.0468	293.711	270.1675	289.1446	287.86
9	294.4102	280.7347	291.5274	290.0561	281.0414	295.9093	282.0241	288.1834	292.0006	281.2214	288.5503	287.6819
10	265.0958	316.2873	270.5311	286.5936	307.0812	259.9539	310.6163	280.5272	275.8961	307.7782	285.4483	287.6388
11	301.1288	270.567	300.6407	285.6425	278.9039	303.5857	272.7661	294.5057	291.7909	277.0224	288.6591	287.7967
12	286.0953	291.2886	284.8397	289.6824	288.2651	285.4759	290.8376	285.3741	288.1763	288.5826	287.8168	287.6612

	M	N	O	P	Q	R	S	T	U	V	W	X
1	287.6843	287.7057	287.706	287.7061	287.7061	287.7061	287.7061	287.7061	287.7061	287.7061	287.7061	287.7061
2	287.7449	287.7019	287.7055	287.7062	287.7061	287.7061	287.7061	287.7061	287.7061	287.7061	287.7061	287.7061
3	287.6572	287.7089	287.7062	287.706	287.7061	287.7061	287.7061	287.7061	287.7061	287.7061	287.7061	287.7061
4	287.7149	287.7031	287.7059	287.7061	287.7061	287.7061	287.7061	287.7061	287.7061	287.7061	287.7061	287.7061
5	287.7414	287.703	287.7055	287.7062	287.7061	287.7061	287.7061	287.7061	287.7061	287.7061	287.7061	287.7061
6	287.6258	287.7101	287.7066	287.706	287.7061	287.7061	287.7061	287.7061	287.7061	287.7061	287.7061	287.7061
7	287.7767	287.7005	287.7052	287.7062	287.7061	287.7061	287.7061	287.7061	287.7061	287.7061	287.7061	287.7061
8	287.6669	287.7065	287.7064	287.706	287.7061	287.7061	287.7061	287.7061	287.7061	287.7061	287.7061	287.7061
9	287.6937	287.7075	287.7059	287.7061	287.7061	287.7061	287.7061	287.7061	287.7061	287.7061	287.7061	287.7061
10	287.7346	287.7024	287.7059	287.7062	287.7061	287.7061	287.7061	287.7061	287.7061	287.7061	287.7061	287.7061
11	287.6818	287.7072	287.7063	287.7061	287.7061	287.7061	287.7061	287.7061	287.7061	287.7061	287.7061	287.7061
12	287.7144	287.7064	287.706	287.7062	287.7061	287.7061	287.7061	287.7061	287.7061	287.7061	287.7061	287.7061

Figure 20: Example of results from file in spreadsheet with conditional formatting.



Earth Simulation

Simulation | Query

Simulation Name:

Axial Tilt: 23.44

Orbital Eccentricity: 0.167

Simulation Time Step
Select value from 1 to 525,600: 1,440

Grid Spacing: 15

Simulation Length in Solar Months: 12

[✓] Check to Enable Display Animation

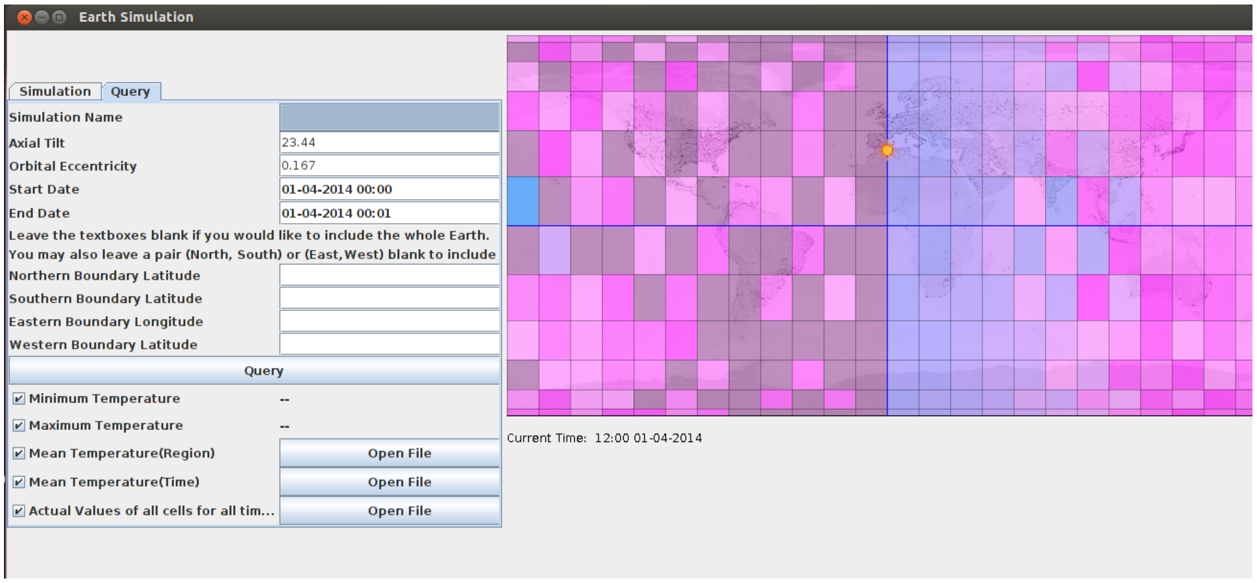
Presentation Display Rate: 5

Start | Pause

Adjust Temp Filter Transparency

Current Time: 12:00 01-04-2014

Figure 21: Initial Heated Planet Graphical User Interface - Simulation Tab



Simulation Name
Axial Tilt 23.44
Orbital Eccentricity 0.167
Start Date 01-04-2014 00:00
End Date 01-04-2014 00:01
Leave the textboxes blank if you would like to include the whole Earth.
You may also leave a pair (North, South) or (East,West) blank to include
Northern Boundary Latitude
Southern Boundary Latitude
Eastern Boundary Longitude
Western Boundary Latitude

☑ Minimum Temperature --
☑ Maximum Temperature --
☑ Mean Temperature(Region) Open File
☑ Mean Temperature(Time) Open File
☑ Actual Values of all cells for all tim... Open File

Current Time: 12:00 01-04-2014

Figure 22: Initial Heated Planet Graphical User Interface - Query Tab

Code Reuse

The presentation layer including many of the gui package components were reused from a prior heated earth project. The image of the Earth and display of the temperatures came from code by Andrew Bernard. Although the domain model from the heated earth project was reused in the user interface for running a simulation, parts of the code were rewritten and enhanced to meet the new functional requirements including planet tilt and orbit.

The overall architecture was based on a Project 2 implementation that facilitated decoupled code and modular programming. The code included the ThreadedProcess, MasterController, SimulationController, PresentationController, SimulationMethod, and PresentationMethod. This code provided a basis for thread management and abstractions of the simulation and presentation layers. It was modified to remove the initiative and threading combinations and parameters were added to meet the new requirements. Reusing this code gave the team a head start on the foundation of the application allowing the members to focus on new development.

Program characteristics

Metric	Value
Lines of code	3,933
Size (in kb)	339
No. of classes	73

Average no. of methods per class	4.78
Average no. of attributes per class	2.21
Number of inter-class dependencies	88

Figure 23: Program Metric Table

Reflection

The design of the project worked well for our team dynamics working in multiple time zones. The architecture allowed for each part of the program to be developed independent of the others therefore allowing each team member to develop out parts at a time. During the implementation we were able to move quickly by reusing code from Project 2 to begin the GUI and some of the calculations. Another strong point of the architecture was the use of the FactoryMethod design pattern which enabled us to use mock objects for simulation result and presentation. Team members could work on the presentation even though the simulation hadn't been completed yet still work with a results class and vice versa.

Creation of simulations and querying the data transparently work well. The user interface may not be the prettiest, but it is effective. We also know the sun does not move north and south which would be expected with a tilted planet, but it does more east to west.

Citations

[1] Gamma, Erich Helm, Richard, Johnson, Ralph and Vlissides, John. Design Patterns: Addison-Wesley, 1995.

[2] Vainolo's Blog [online] Available - http://www.vainolo.com/2012/05/02/factory-method-design-pattern-uml-modeling/

[3] Vainolo's Blog [online] Available - http://www.vainolo.com/2012/04/29/singleton-design-pattern-sequence-diagram/

[4] Core J2EE Patterns - Data Access Object [online] Available - http://www.oracle.com/technetwork/java/dataaccessobject-138824.html

Design Patterns

FactoryMethod

[1] Gamma, Erich Helm, Richard, Johnson, Ralph and Vlissides, John. Design Patterns: Addison-Wesley, 1995.

The FactoryMethod is used to allow support for mock objects used for testing purposes and the ability to support multiple different data persistence methods such as databases, flat files or other. The pattern allows the creation of the appropriate logical component without the consuming component to have knowledge of the concrete implementation. Thus the consumer may act without knowledge of the implementation details.

PlanetSim leveraged this pattern to abstract the creation of mock test objects. The SimulationController when run, calls the getSimulationDAO method in Objectfactory which calls the SimulationDAOFactory (Creator implementing SimulationDAO interface) and returns a type of subclass either SimulationRDBMSDAOFactoryImpl or MockSimulationDAOFactory.

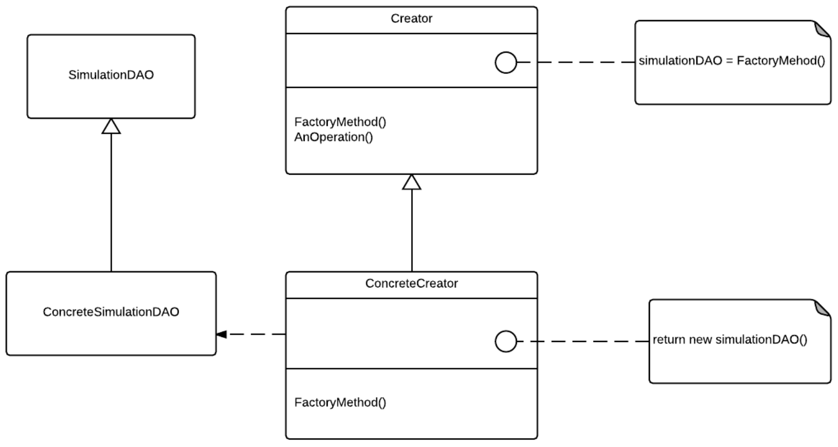


Figure 24: FactoryMethod Class Diagram

The Participants:

- *Product*: the interface of objects that the Factory Method creates.
- *Concrete Product*: responsible for implementing the product interface.
- *Creator:* determine the Factory Method and returns an object of that type.
- *Concrete Creator:* overrides the Factory Method and returns an instance of concrete product.

In the class diagram described in Fig.24, the creator relies on its subclasses to determine the Factory Method in order to return an instance of the correct concrete product.

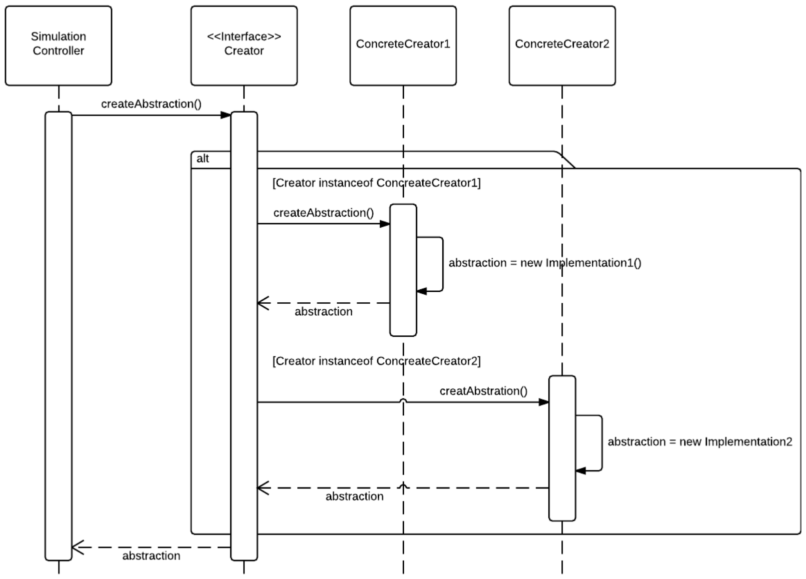


Figure 25: FactoryMethod Sequence Diagram

The sequence diagram illustrated in Figure x demonstrates the flow of operations within the Factory Method design pattern. The client is unaware of how the pattern determines which concrete object is returned as that is determined completely by the sub classes.

Singleton

[1] Gamma, Erich Helm, Richard, Johnson, Ralph and Vlissides, John. Design Patterns: Addison-Wesley, 1995.

The Singleton design pattern is used in a few spots one is to create a single instance of a class responsible for connecting to the database. In order to manage the existence of multiple connections to the database, a singleton pattern was chosen to control creation of the underlying datastore.

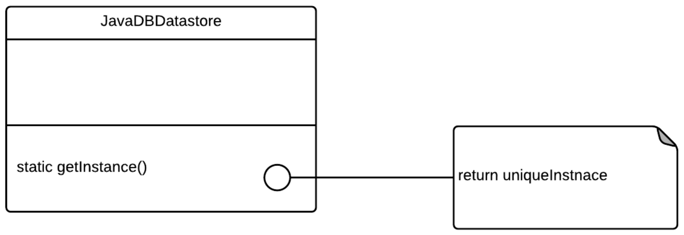


Figure 25: Singleton Class Diagram

The Participants:

- *Singleton*: Responsible for creating its own unique static instance.

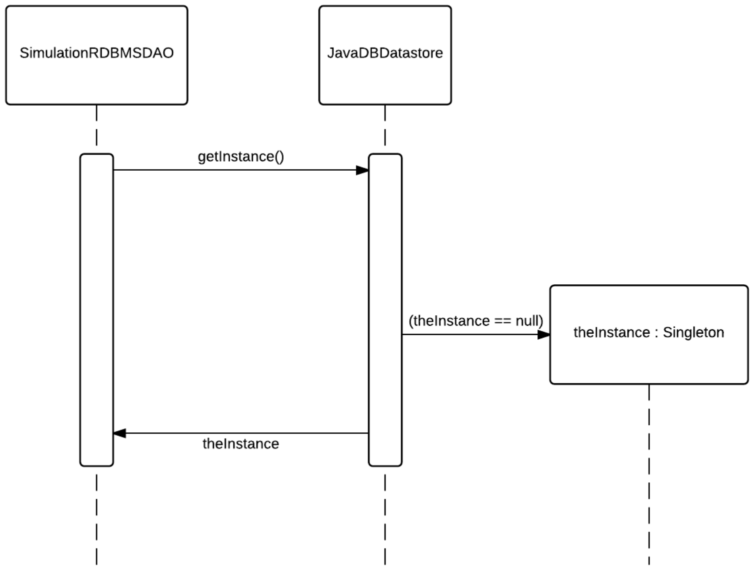


Figure 26: Singleton Class Diagram

AbstractFactory

[1] Gamma, Erich Helm, Richard, Johnson, Ralph and Vlissides, John. Design Patterns: Addison-Wesley, 1995.

The AbstractFactory pattern is used to provide an interface to families of related or dependent objects without specifying their concrete classes. Particularly, this was useful in abstracting the persistence code for the simulation program in order to easily switch out the underlying datastore. The decision to use this pattern was based on to needs. One the decision to utilize a relational database was not an made with absolute confidence so the option to switch to another data store without impacting coding on the rest of the system was beneficial. Two the abstraction provided an easy mechanism to inject mock persistent components for testing

.

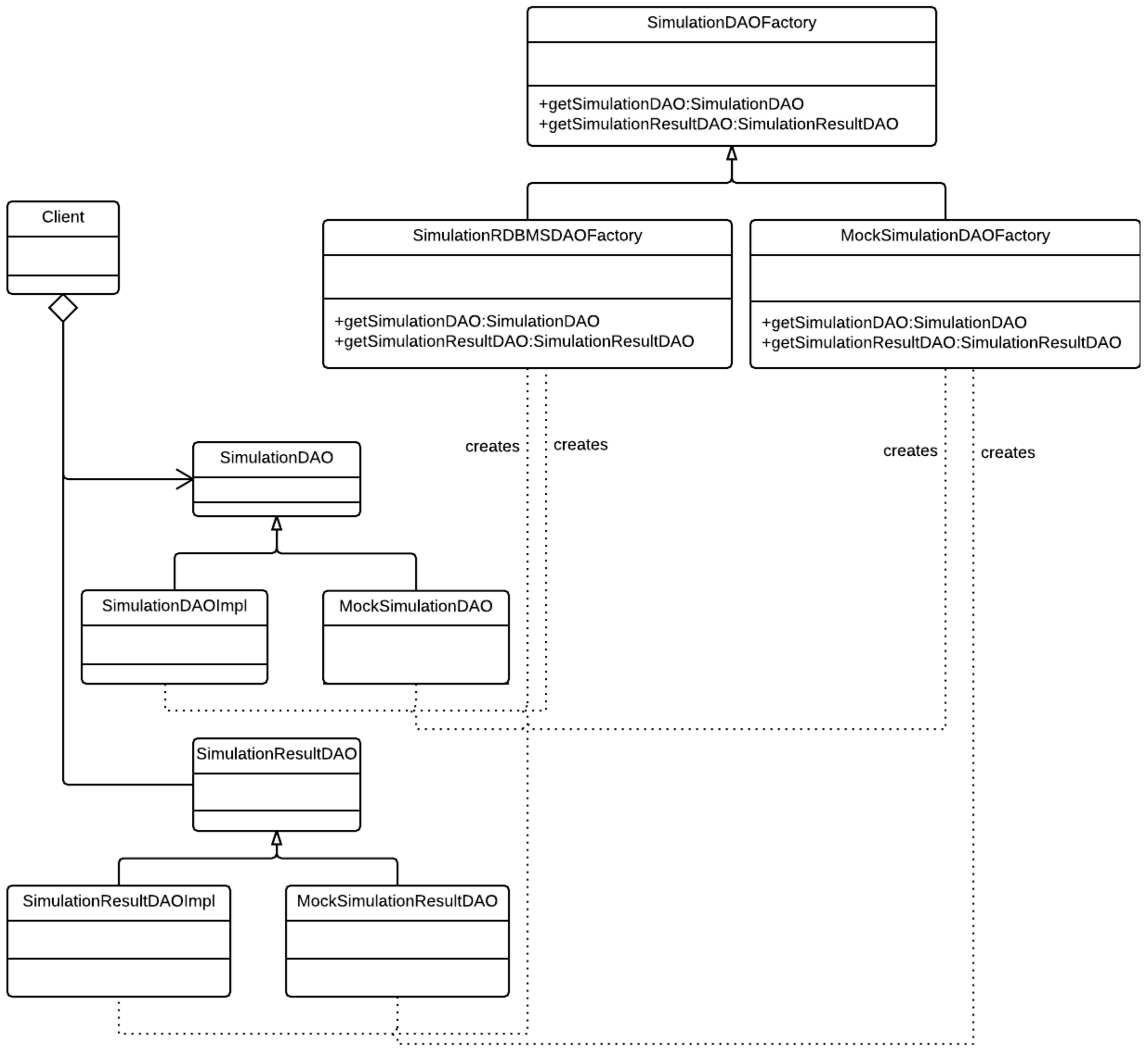

SimulationDAOFactory
+getSimulationDAO:SimulationDAO
+getSimulationResultDAO:SimulationResultDAO
Client
SimulationRDBMSDAOFactory
+getSimulationDAO:SimulationDAO
+getSimulationResultDAO:SimulationResultDAO
MockSimulationDAOFactory
+getSimulationDAO:SimulationDAO
+getSimulationResultDAO:SimulationResultDAO
creates
creates
creates
creates
SimulationDAO
SimulationDAOImpl
MockSimulationDAO
SimulationResultDAO
SimulationResultDAOImpl
MockSimulationResultDAO

Figure 27: Abstract Factory Class Diagram

The Participants:

- **AbstractFactory:** declares the interface with operations that create abstract products.
- **ConcreteFactory:** implements the AbstractFactory class and create concrete projects.

- **AbstractProduct**: declares an interface for a type of product objects from the AbstractFactory operations.
- **Product:** implements the corresponding AbstractProduct interface and specifies a product to be created by the ConcreteFactory.
- **Client:** uses AbstractFactory to create objects needed which are returned as abstractions of the concrete objects created

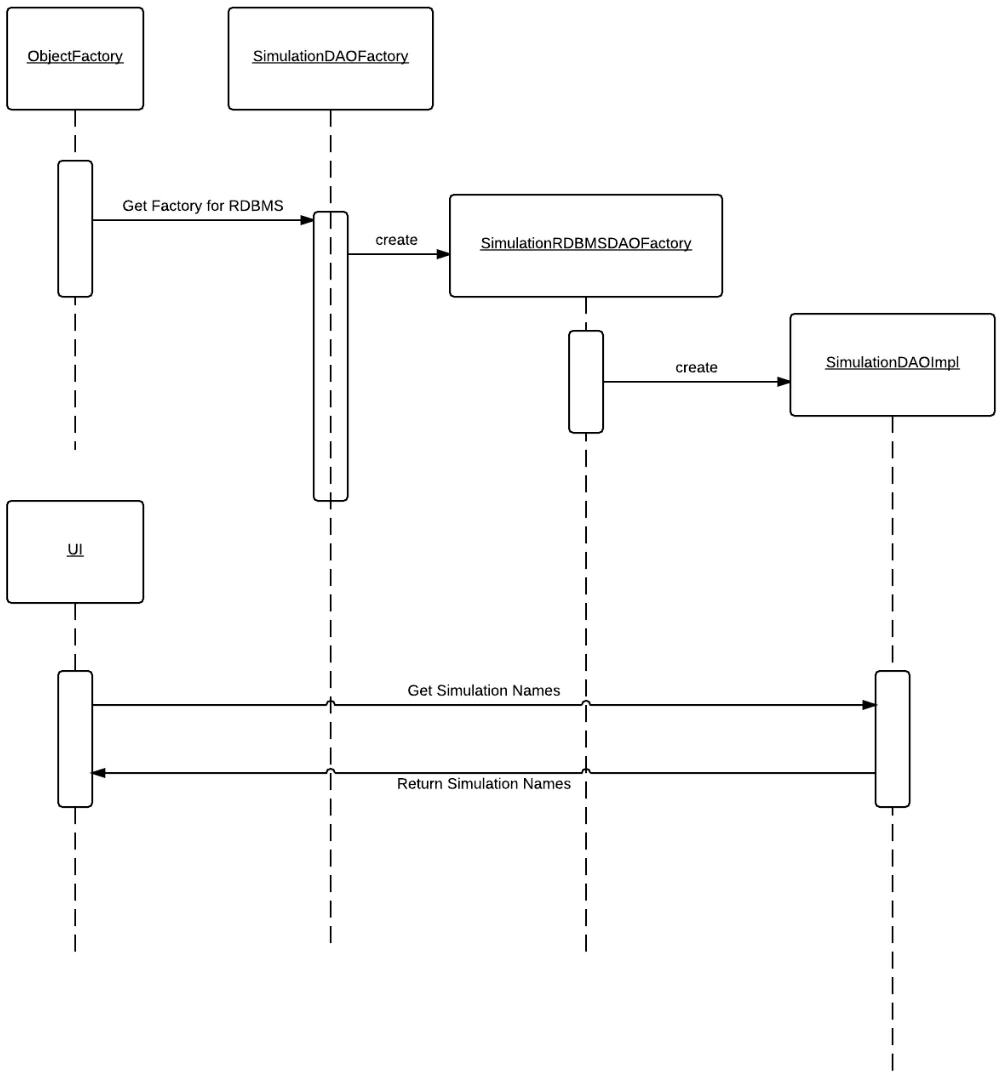


Figure 28: Abstract Factory Implementation Class Diagram

Data Access Object (DAO)

[4] Core J2EE Patterns - Data Access Object [online] Available - http://www.oracle.com/technetwork/java/dataaccessobject-138824.html

The Data Access Object pattern is used to control the serialization of objects into a database for persistence. The DAO pattern abstracts and encapsulates the underlying persistence mechanism of an object. It acts primarily as an access mechanism, retrieving persisted object from the database. In our program, it was especially useful in order to store and retrieve simulations without maintaining SQL queries.

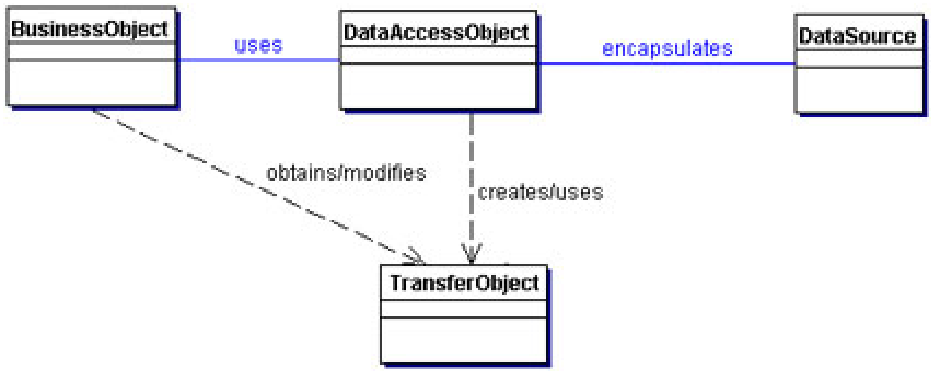


Figure 29: DAO Generic Class Model Diagram

In our class model diagram, the business object are the UI and the SimulationController. They use DataAccessObject such as the the CellDAO and SimulationDAO, which connect to the Data Source; the JavaDBDataStore class.

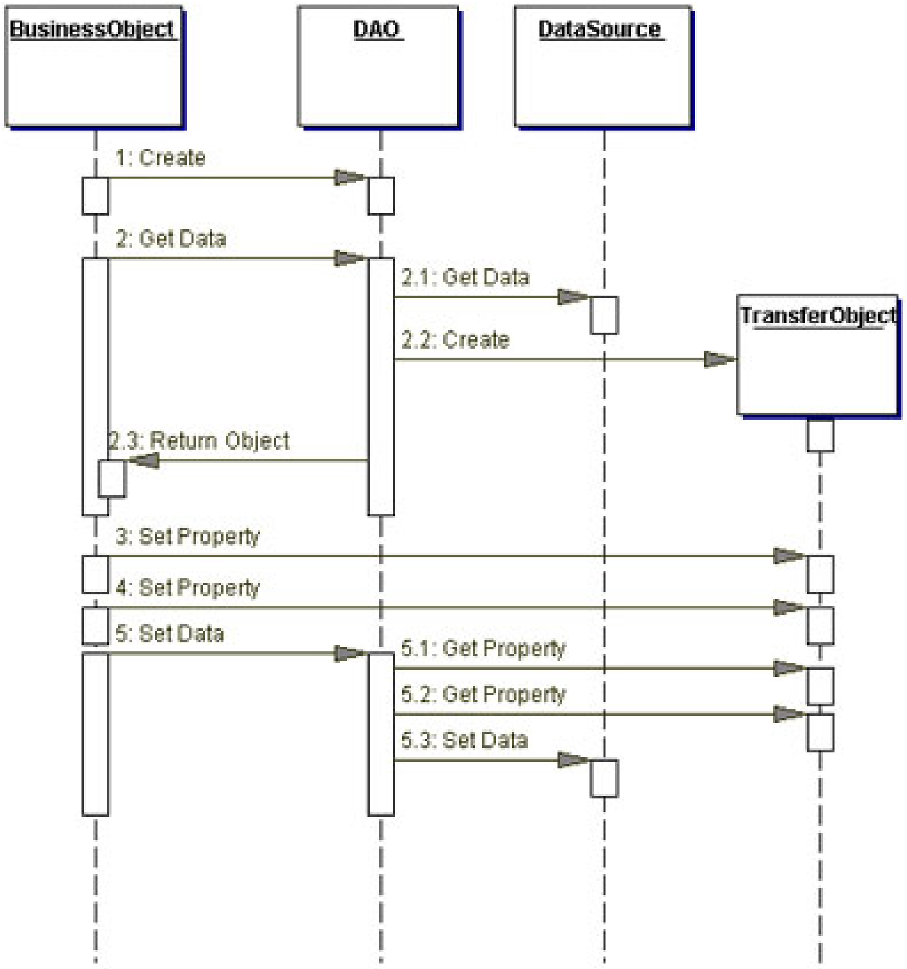


Figure 30: DAO generic sequence Diagram