AN APPROACH TO VISUAL MODELING OF CELLULAR AUTOMATA

by

Sajjan Sarkar

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

ARIZONA STATE UNIVERSITY

August 2009

AN APPROACH TO VISUAL MODELING OF CELLULAR AUTOMATA

by

Sajjan Sarkar

has been approved

April 2009

ABSTRACT

Cellular Automata (CA) are important to scientific and engineering fields with applications in studying landscape erosion dynamics, fire spread patterns, and disaster relief planning. Visual modeling tools are highly desirable for creating component-based simulation models of complex systems. Such tools can significantly simplify model development tasks such as creating models for the individual components of a system and synthesizing them to represent different models of the system. However, the current approaches and available tools for developing cellular automata models are restrictive given that they depend on pre-built models or require developing code. In this thesis, a novel visual modeling tool called CoSMoS-CA is developed. It supports CA-specific model development by extending the Component-based System Modeler and Simulator (CoSMoS) framework and using a spatial-specialization relationship between a CA and one or more sets of the CA's cells. The resulting CoSMoS-CA modeling tool enables flexible creation and manipulation of cellular automata models and supports automatic model instantiations. The main benefits of the visual modeling tool are exemplified using a basic landscape model that undergoes a simple erosion process.

To My Family

ACKNOWLEDGMENTS

First of all I am deeply grateful to my thesis advisor Dr. Hessam Sarjoughian, for introducing me to the science of Modeling and Simulation and hence making this thesis possible. I would like to thank him not only for his vast knowledge and expertise in the field, but also for his unwavering support, kindness and constant guidance through every step of not only this thesis and research but also in my association with him as his teaching assistant. To be able to work with a person of his integrity and thirst for perfection has been a tremendous experience.

I would also like to thank my thesis committee, Dr. Aviral Shrivastava and Dr. Dijiang Huang for their time and efforts in reviewing this thesis.

My next thanks go to Gary Mayer, my colleague and friend, who was always ready to help me understand my research by offering valuable suggestions, insights, cookies and ideas.

I am grateful to all the members at ASU-ACIMS (Arizona Center for Integrative Modeling and Simulation), for their help in discussions on my research, their support and friendship.

I would also like to thank my friends, both here and in India for their encouragement and good wishes.

Finally, my deepest gratitude goes to my family, for their unflinching support, love and blessings. When things look hopeless, it is to them I turn and find my strength.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF LISTINGS

LIST OF FIGURES

Figure Page

Figure Page

CHAPTER 1

INTRODUCTION

1.1 Research Objectives

Cellular Automata structures are complex to visualize when seen as component-based constructs [3]. On one hand, *every rule* that applies to a hierarchical component-based system, for example, closure under coupling and modularity *must be* followed by the CA, thereby making the CA a "type-of" (albeit much constrained) component-based system. On the other hand, the CA in turn enforces rules of its own which, when combined with the rules of the component-based system seem to constrain it so much that it can be argued that CAs must be treated as highly conditional specializations of component-based system. Some of these constraining rules are part of a CA's basic definition. An example could be that a CA must comprise of cells which must be identical to every other cell in terms of logical structure and behavior specification. The only difference between one cell and another is its state at a given time instance and its location with respect to the CA's coordinate.

Alternative approaches exist for modeling CAs. A Cellular Automata model may be specified in terms of general-purpose or specialized programming languages, a modeling formalism, or their combination. Specific execution or simulation algorithms are used to reveal the dynamics of the CA models.

Apart from the complexity involved with modeling, Cellular Automata present unique constraints (e.g., managing large-scale models and creating different instances of the models) and advantages as compared with generic component-based models. Therefore, the goal of this research is to take advantage of the inherent simplicity of CA models and making CA model development accessible to individuals who may have limited knowledge and experience in computer programming languages.

1.2 Contribution

The contributions of this thesis can be summarized as

Developed CoSMoS-CA design and implementation which enables visual CA modeling using the concept of *spatial specialization* which is important for initialization of the CA's cells in a hierarchical fashion. The CoSMoS-CA environment supports model development, initialization, and generation of models that can be simulated using cellular DEVS [16].

Defined a process where model development for cellular automata can be carried out systematically.

An example CA model is developed in CoSMoS-CA. The basic modeling steps in developing CA structural model specification are shown using a simple dynamic landscape soil erosion model and how the model can be automatically translated for execution using the Cellular DEVS simulation engine [1].

1.3 Organization of the Thesis

Chapter 2 presents an overview on cellular automata including network structure and rules that govern their dynamics. Next a description of the Component-based System Modeling and Simulation (CoSMoS) framework [2,4,6,9,10,11] is given and its overall capabilities and features are discussed including the types of models that can be created and simulated and tracked on the built-in DEVS-Suite [5]. Then we present and compare 3 commonly used modeling environments NetLogo [13], Mathematica [15] and Cellular DEVS [16]. Chapter 3 discusses how cellular automata may be viewed as having properties similar to component-based models and discusses the different semantics like handling edge conditions and applying component-based modeling concepts like Specialization and composition to cellular automata. Chapter 4 describes the design and implementation of the CoSMoS-CA environment [4]. Software design specifications including flow charts, class and sequence diagrams are developed and described. Chapter 5 introduces an exemplar model in which soil erosion of a mountain slope is modeled in which a set of models are developed step-by-step in order to show the capabilities of the CoSMoS-CA. Chapter 6 presents conclusions and discussed future research.

CHAPTER 2

BACKGROUND

2.1 Cellular Automata

A cellular automaton or CA can be considered as a mathematical "machine" or an "organism" that lends itself to some very interesting dynamics. Though they are remarkably simple at the start, CAs has a variety of applications like landscape modeling, population pattern analysis, and cryptography. Cellular automata are simple mathematical idealizations of natural systems. For example, a CA can consist of a lattice of discrete identical sites (e.g., a village or a farm), each site taking on a finite set of, say, and integer values. The values of the sites evolve in discrete time steps according to rules that specify the value of each site in terms of the values of neighboring sites. There is an implied spatial relationship, and complex behaviors can be derived from simplistic rules.

Each CA is defined as a set of interconnected homogeneous cells connected using a network with a particular connectivity pattern. Every cell is exactly identical to every other cell in terms of spatial dimensions, shape, attributes and behavior.

This research attempts to focus on the structure and behavior of a CA when viewed as a component-based model from which families of hierarchical models may be constructed. Our goal is to show that cellular automata are a class of problems that may be developed visually using component-based modeling paradigm. In order to use well-defined component based design principles we use the Component based System Modeler and Simulation framework.

2.2 CoSMoS (Component-based System Modeling and Simulation)

CoSMoS (Component-based System Modeling and Simulation) is a modeling environment aimed at visual component-based simulation model specification [6] [12]. Figure 1 shows the CoSMoS modeling environment. CoSMoS provides a unified framework in which models have distinct logical, visual, and persistent specifications. It supports component-based modeling approaches such as DEVS and XML [10]. CoSMoS also supports a class of Non-Simulatable Model (NSM) [11] [12] components. These kinds of models are based on object-oriented model components. They are depicted differently than the Simulatable Model (SM) primitive and composite components which are time-based. One of the main differences between SM and NSM is that the execution of the SM model components is determined by the simulation protocol.

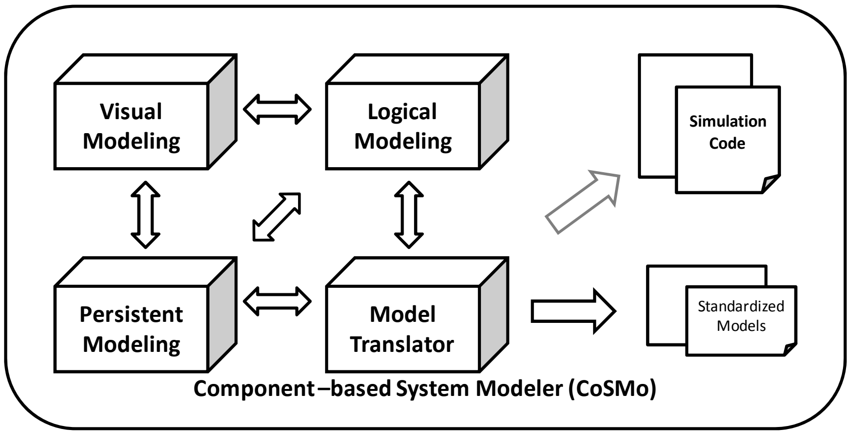


Figure 1. Logical, visual, and persistent model types with model translators

The logical model specification is defined for primitive and composite models. A set of axioms that ensure consistency of alternative hierarchical model specifications is provided. CoSMoS provides the facility to design the models at different levels of resolution by using the separation of the Template Models (TM), Instance Template Models (ITM) and Instance Models (IM) [6][12] as depicted in Figure 1. A template model (TM) specifies primitive and composite models as components with input and output ports and values. The modeler may also specify values for the state variables of the cells and the specializations. The ITM model is defined to have a finite hierarchy of depth greater than two. Furthermore, this model may not specify multiplicity of a model component within a composite model - a model can have one to a finite number of copies of the same instance template model. An instance model is an instantiation of an instance template model where the multiplicity of model instances. The types and values for input and output can also be specified [2]. The translator provides the transformation of the data in the database to runnable simulation models. The data types of the input and output ports of primitive and composite models can be specified as well as behavioral and structural metrics [9]. CoSMoS (Component based System Modeler and Simulator) extends the capabilities of CoSMo by integrating it with the DEVS-Suite environment which supports simulating the generated models with animation, TimeView, and Tracking Log [7]. The basic aim is to reduce the model development cost significantly, lower the learning curve for general users, and improve the comprehension of continuously evolving models, making them suitable for efficient decision making [14].

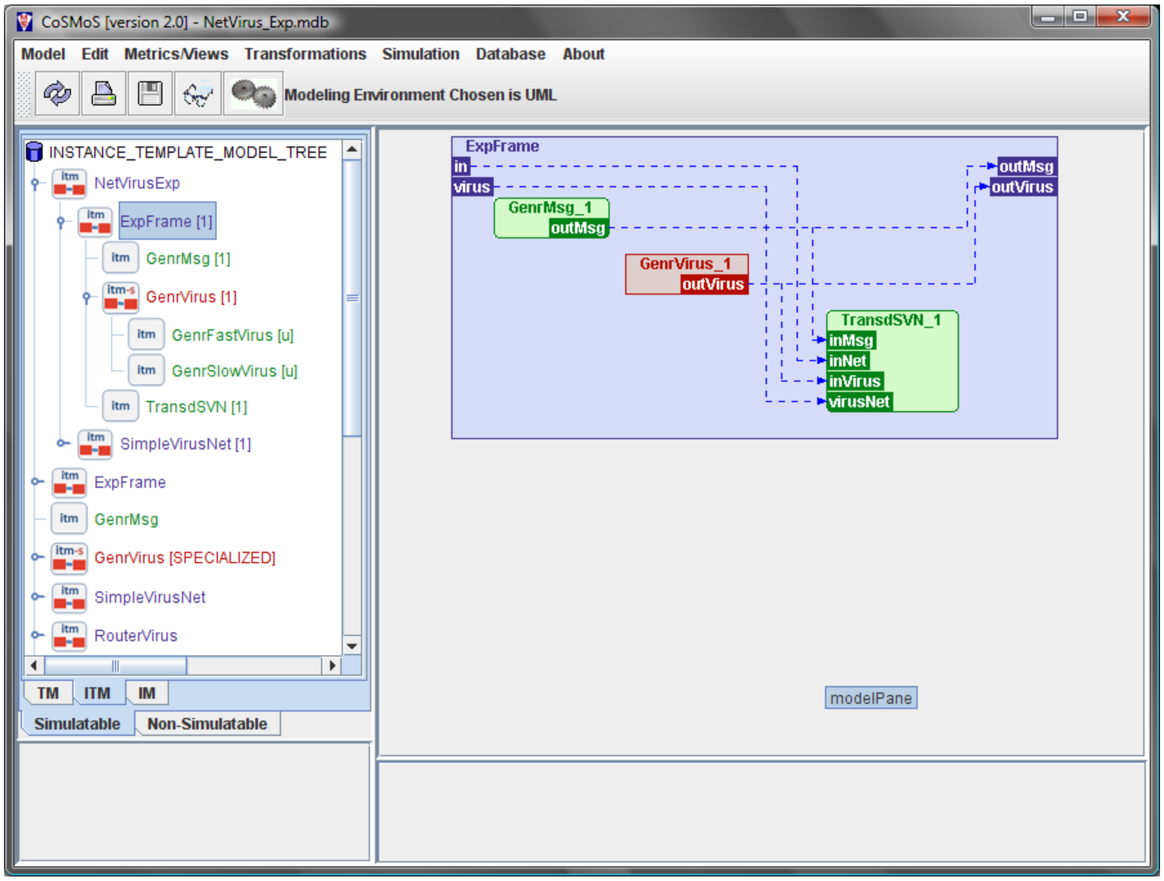


Figure 2. CoSMoS User Interface

The graphical user interface used for visual modeling is a key component and provides three complementary definitions of every model which are the Template model (TM), Instance Template Model (ITM) and then the Instance Model (IM) described above. The models are visualized in 2 views, one shows the hierarchical composition using the Tree structure that lists all the primitive and composite models and their parts and specializations and the other is the graphical block representation that shows the primitive models and the composite models up to two levels of hierarchy. These models may be exported to simulation code like Java or XML. For Java, the models conform to the DEVSJAVA API specification. These partial

DEVSJAVA source code generated for each atomic model can be completed by providing the implementation of the external, internal, output, and time advance function using any IDE's or the editor that is provided with CoSMoS. The models inside the database of CoSMoS can be translated into partial simulation code which can be run by the DEVS-Suite simulation engine which is integrated into CoSMoS [11]. CoSMoS uses a database to persist its models and, coupled with a translator allows the modeler to build complex and multi-component systems.

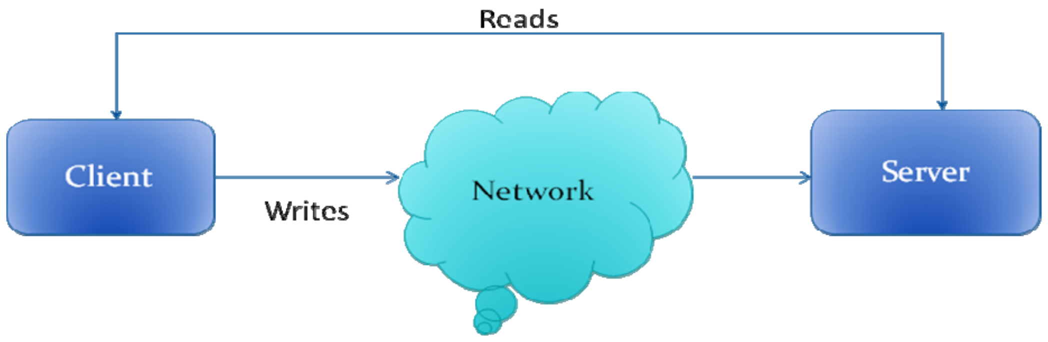


Figure 3. Client-Server Architecture Concept for CoSMoS

The basic architecture of the CoSMoS is client-server. The main parts of the software are Client, Network, and Server. The Client requests for the write requests which are managed by the Network and then processed by the Server ensuring concurrency. The server also enforces some rules according to the CoSMoS's axioms in order to maintain the syntactical correctness of the models. Read operations do not go through the network but are called independently.

CoSMoS create easy-to-understand visual representations that can reduce the effort and cost of the entire simulation model creation lifecycle. To our knowledge, no modeling tool offers the capability to visually create a family of CA models as in CoSMoS.

2.3 Related Work

2.3.1 Evaluation Metrics

We compare CoSMoS-CA with 3 common tools NetLogo [13] and Mathematica7 [15]

We broadly divide the evaluation metrics into 2 aspects, visual modeling and logical modeling.

Visual Modeling:

Visual Modeling Capability: This evaluates whether the tool supports construction of complex cellular automata using visual modeling.

Logical Modeling:

Support for Cellular Automata: Whether the tool supports cellular automata modeling.

Reusable Component-based: This evaluates if the tool allows the user to create models, save them and reuse them as components while creating other models using part-of relationships.

User-Defined Rules: Evaluates if the modeler can create custom models and include complex behavior applicable for his/her domain.

Specialization: Whether the tool allows a CA to have cells with different initialization values for its state variables. For modeling real world entities this is very important as this allows us to create spatially specialized regions.

Multi-Dimensional CAs: Evaluates if the tool supports cellular automata with more than 2 spatial dimensions.

2.3.2 Cellular DEVS

Cellular DEVS is an extension of the DEVS formalism which is used to create models of cellular automata. Cell-DEVS models defined as a space composed of individual cells. The cells are defined as atomic models that can be lately coupled to form a complete CA model. It is built using the spatial multi-component DTSS described in [16]. One of its implementations is its Cellular DEVSJAVA which is a Java API that extends the DEVSJAVA API by providing classes to create CA models and simulate them. Cellular DEVSJAVA is primarily code-based, meaning models have to be built programmatically which requires the modeler to be familiar with the DEVS and Cellular DEVS formalisms and does not support visual model construction.

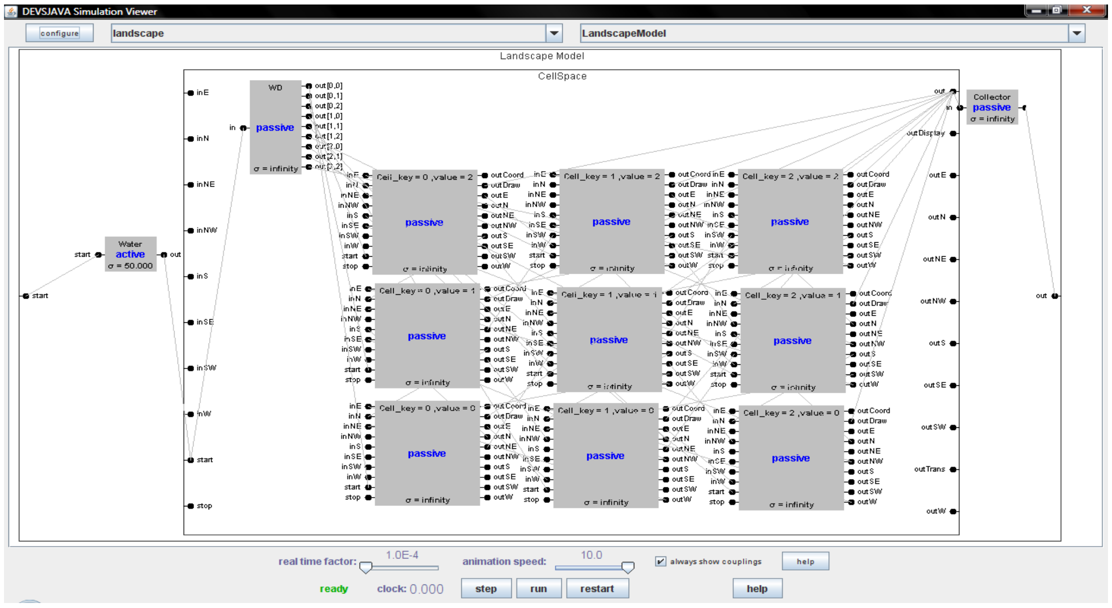
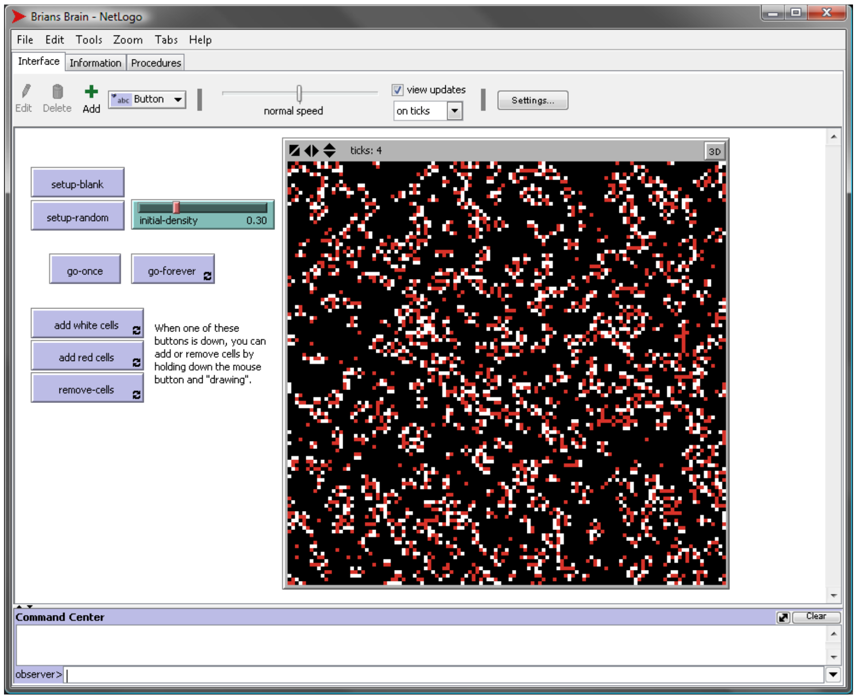


Figure 4. Cellular DEVS user interface

2.3.3 NetLogo

NetLogo is a cross-platform multi-agent programmable modeling environment aimed at modeling complex models that change over time. It has support for developing cellular automata models from pre-built templates. It has a visual interface where the user can seed cells and view simulation runs. Simple rules for the cells may be created using the native NetLogo programming language. While NetLogo allows the user to visually seed cells, it doesn't support creating regions in the CA with different initialization values. All cells have to either dead or initialized with homogeneous values. Furthermore, there is no support to model cells as first class objects with explicit representation of inputs and outputs and thus coupling of cells to form CAs. The code is needed to render visualization of a CA.

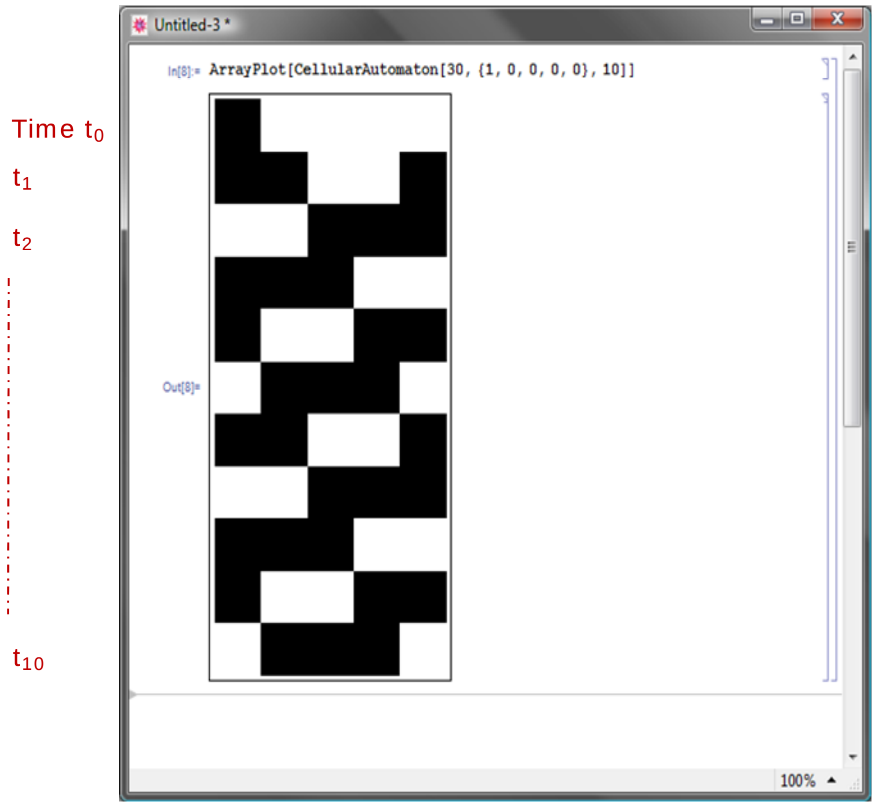


When one of these buttons is down, you can add or remove cells by holding down the mouse button and "drawing".

Figure 5. Cellular Automata user interface in NetLogo

2.3.4 Mathematica

Mathematica is a multi discipline computational software program that also does model building using interpreted expressions. It has support for Cellular Automata including multi-dimensional cellular automata. Mathematica is split into two parts, the "kernel" and the "front end". The kernel interprets expressions (Mathematica code) and returns result expressions. The Front End provides a document-centered GUI. But visual GUI-based model building is not supported, as the GUI only shows the output of evaluated expressions. Mathematica does not allow user defined rules for Cellular Automaton.



```
ArrayPlot[CellularAutomaton[30, {1, 0, 0, 0, 0}, 10]]
```

Figure 6. CA Model in Mathematica

Table1 is a comparison of the above approaches and tools. As noted above, there are basic differences between visual modeling that is supported for the proposed CoSMoS-CA and other tools such as Netlogo.

Tool	Visual Modeling		Logical Modeling				Model Translation	Persistence Model	Memory Footprint
	Viewing	Creating	Families of Models	Visual Specialization	User-Defined Rules	Multi-Dimensional (>2D)			
Cellular DEVS	yes	no	no	yes	yes	yes	n.a	Flat file (Java .class files)	32.8 MB
NetLogo	yes	yes	no	yes	yes	no	no	Flat file (.nlogo)	38.8 MB
Mathematica 7	yes	no	no	no	no	yes	n.a	Flat file (.nb)	12.7 MB
CoSMoS-CA	yes	yes	yes	yes	yes	no	yes (Java)	Database (MS-Access)	31.1 MB

Table 1. Comparison of, Cellular DEVS, NetLogo, Mathematica and CoSMoS-CA

As can be seen from the table above, CoSMoS-CA supports creating families of models as will be discussed later. NetLogo allows the modeler to define a set of homogeneous cells, each of which is initialized differently. These cells must be specified and initialized by writing code before the CA is loaded in the GUI. These initialization values cannot be created, modified or deleted from the GUI. This results in a set of pre-defined specialized cells which appear as options in the GUI. When the modeler wants to specialize cells in the CA to have initialization values identical to one of the pre-defined specialized cells, the modeler can select the corresponding GUI option for that pre-defined cell and then click on the cell in the CA which he wants to initialize as per the pre-defined values. The selected cell in the CA is assigned the same initialization values as the pre-defined cell represented by the selected GUI option. In contrast to CoSMo-CA and NetLogo, Mathematica supports visualizing simulation outputs in two-dimensional grids.

CHAPTER 3

APPROACH FOR VISUAL CA MODEL CONSTRUCTION

3.1 Network Connectivity Patterns

This is the first aspect of the process of modeling and simulation. This involves conceptualizing and building of models. The modeling environment allows visual creation, deletion, copy, modification of CAs and their couplings. Here copying means to be able to create a new model with same properties except different name.

One of the goals of this research is to be able to provide a platform for visually creating CAs visually using a GUI-based approach similar to the environment provided by CoSMoS. This could involve features like having pre-built cell templates and manipulating their specifications.

One of the aims of this research is to show that visual modeling of CAs is more user-friendly and can be done efficiently with minimal programming and thus making modeling more accessible. While the modeler would still have the full freedom to enforce application-specific behavior and states, the system would abstract the modeler from implementing the entire model by providing automated processes for cell replication in the CA, implementing the connectivity pattern and creating code stubs for the models.

Cellular Automata (especially ones that have 2 or more dimensions) can have fairly complicated couplings among neighboring cells. Two common neighborhood connectivity patterns are the Moore and the Von Neumann.

Neighborhood Connectivity Patterns:

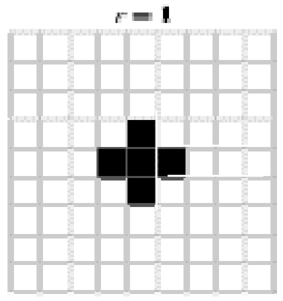


Figure 7. Von Neumann Neighborhood

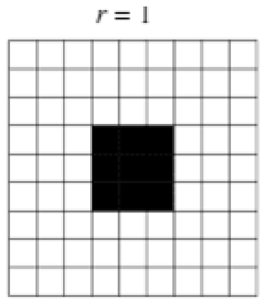


Figure 8. Moore Neighborhood

The von Neumann neighborhood comprises of four cells orthogonally surrounding a central cell on a two-dimensional square lattice. The Moore neighborhood comprises the eight cells surrounding a central cell on a two-dimensional square lattice.

3.2 Edge Conditions

A special mention needs to be made about edge conditions, i.e. cells on the edge of the cellspace. They need to be carefully handled as they could easily change the state of the entire CA. In theory, CA is infinite and has no boundaries. Every cell has neighbors to whom it passes its output. But in real world it is not feasible and practical to model infinite spaces, so boundaries are applied to it, creating edges. The cells at the outer periphery of the CA are called edge-cells. One of the problems that these boundaries bring is that edge-cells have different structure as they have fewer neighbors. Therefore, it is important to be precise about handling of the boundary cells.

The simplest (and least optimal) solution is to just ignore the outputs of the edge cells. This approach could have serious repercussions on the overall output of the CA as the information from these cells is not saved in the system and the system behaves differently because of its absence. Consider for example a 5 x5 Moore connected CA. Every cell has exactly the same rule set. For example we could consider every cell to have the rule for Conway's Game of Life. The rules defined by Conway are as follows [3]

A live cell remains alive if it has between 2 or 3 live cells in its neighborhood.

A live cell will die of overcrowding if it has more than 3 live cells in its neighborhood

A live cell will die of isolation if it has less than 2 live cells in its neighborhood.

A dead cell will become alive if it has exactly 3 live neighbors.

However, although in theory this is not the most optimal solution, it is the only practical way to implement CAs and presents lesser complexity. Another approach could be to enforce different behavior for the edge cells. This approach comes with its own set of considerations like understanding that this would mean that the CA is no longer homogenous but is composed of different "kinds" of cells. Also making sure that the edge rules are in coherence with the overall CA's behavior could be tricky. A third approach could be to treat the grid as a virtual torus in which outputs from the edge cells on one side of the CA become inputs to the cells on the other side of the grid. This may not be applicable to all systems due to its closed nature and continuous looping could be undesirable from a domain perspective. The focus of this research will not explore the different aspects of boundary cells in CAs. The future objective is to provide a way to build simple CAs which eventually allows the user to model a heterogeneous multi-component system in which one or more components could be a CA. These types of CAs are called Composable CAs [8].

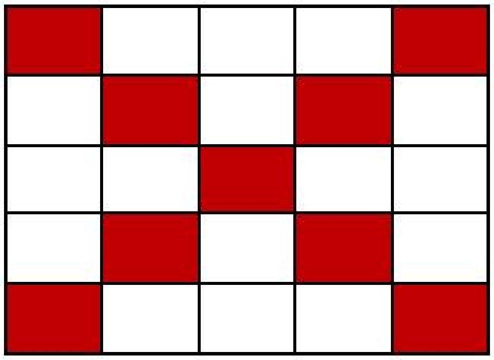


Figure 9. Initial CA

non-Torus after state change .Torus after state change

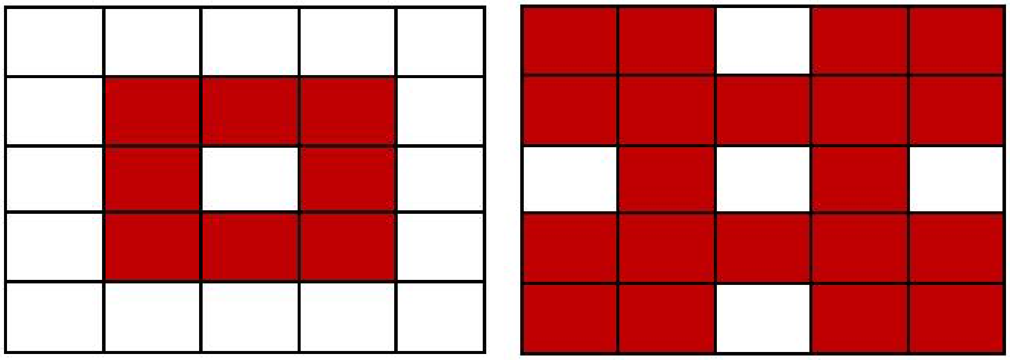


Figure 10. Difference in pattern for Torus and non-Torus CAs

3.3 Semantics of Cellular Automata in CoSMoS

3.3.1 The Cell

The cell is the base model for creating a class of CAs. It is the fundamental, atomic unit of a cellular automaton network. In pure CA, a cell would have:

1. States, where each state has:
    a. name-String
    b. type-values could be integers, string, floats etc
    c. initial value- values could be integers, string, floats etc

2. Behavior:
    a. These are rules which determine the output of the cell when it receives an external input. Typically the new state depends on the old state, the input and the rule.
3. Shape and dimension
    a. In theory, CAs could have arbitrary number of dimensions, typically 1D, 2D and 3D CAs are most common. The cells themselves could be finite n-dimensional shape except shapes such as oval, circles etc. For this research, only square cells are modeled.
4. Inducers
    a. These are the set of cells which can influence the state of the cell.

b. In CoSMoS, we maintain this information at the CA level based on the network connectivity pattern.

We add the port construct to represent and simulate the state change information transfer between influencer and influence.

Input port: this is used to receive the information from an influencer that would cause the influencee to change state based on the rules.

Output port: this is used to inform the cell's influence about the influencer's state change.

The state change information is passed to the influencees using messages sent from the influencer's outport into the influencee's in port.

3.3.2 The CA

While constructing a network of cells we first define a CA Model which is essentially the representation of the spatial couplings of the cells. So the CA receive no external input and output. The CA model only has knowledge of the spatial dimensions of the CA(e.g. 2D), its size(e.g. 20 x 10),its coordinates (e.g. {0,0}), its network connectivity pattern and its name

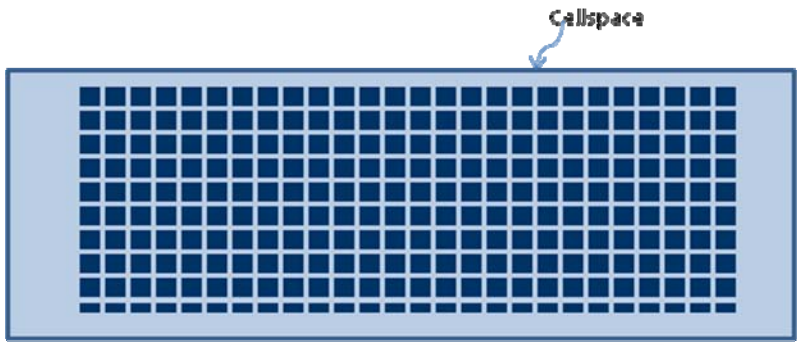


Figure 11. Example of a 26 x 92 two-dimensional CA

We define the CA as a spatial multi-component DTSS model. [16]

MultiDTSS = <D, {M_{ij}}, h_n> where,

D = is the index set

$M_{i,j}$=set cells where $M_{i,j}$=< $Q_{i,j}$,$Y_{i,j}$,$I_{i,j}$, $\delta_{I,j}$,λ>

Where,

$Q_{i,j}$: State

Y=output set (empty)

I=set of influencers

$\delta_{I,j}$= state transition function

λ= overall output (empty)

h_n=time base

When creating a CA the user needs to specify properties for the cell as well as the CA as a whole.

Properties of the CA Model are:

It has no External Internal Coupling (EIC) or Internal External Coupling (EOC).

The network metadata like spatial dimensions, size, coordinates are maintained in the CAModel.

It has no state variables of its own.

All cells subsumed by the cellspace must be homogeneous and must be coupled uniformly. All cells except edge cells have complete connectivity to their neighboring cells. Each edge cell has a partial set of neighboring cells.

3.3.3 DEVS Formalism for CA

Using the DEVS Formalism [16],

$N_{Cellspace}$=<X, Y, D, {M_d| d Є D}, EIC, EOC, IC>

Inports= {Ø}

Outport= {Ø}

X =real valued input set

Y = {Ø}

D= {cell[0,0],cell[0,1]……………..cell[x,y]} where x,y are the CA dimensions.

$M_{cell[0,0]}$= $M_{cell[0,1]}$= $M_{cell[0,2]}$=………………..= $M_{cell[x,y]}$=cell

EIC= {Ø}

EOC= {Ø}

IC= {((Cell [i, j], outE), (Cell [i+1, j], inW)), ((Cell [i, j], outW), (Cell [i-1,j],inE)),

((Cell [i, j], outN), (Cell [i,j+1],inS)),

((Cell [i, j], outS), (Cell [i, j-1], inN)),

((Cell [i, j], outSE), (Cell [i+1,j-1],inNW))}

((Cell [i, j], outSW), (Cell [i-1,j-1],inNE)),

((Cell [i, j], outNE), (Cell [i+1,j+1],inSW)),

((Cell [i, j], outNW), (Cell [i-1,j+1],inSE))

The system would allow the user to inject initialization values across the whole CA or allow the user to specifically go to a particular cell in the CA and initialize it differently. The capability to zoom into and out of the cellspace to look for a particular cell or a set of cells would be provided. While the actual ports and port connectivity would be done by the system and not shown by default, the user would still be allowed to select which port needs to be tracked (data collected for simulation visualization). A scalable mechanism needs to be developed.

The cells would have ports which would enforce the network connectivity model which the user had specified earlier.

The behavior of the cells would be implemented by the modeler. The system would allow exporting the model into one or more specific simulation language (e.g., Cellular DEVS realization in Java) so that the user can implement the behavior of the cells. Using the Cosmo's support for adding behavior to models there would be no need to develop new code implementation. Such a model could be simulated in the DEVS-Suite simulator. Once simulation begins, the user can see the values in the cells that he has chosen to track. This could be visualized on CoSMoS' UI viewers.

3.4 CAs as CoSMoS Models

Instance Template Model:

In CoSMoS, this type of model is defined to have a finite hierarchy of depth greater than two. Furthermore, this model specifies multiplicity of a model component within a composite model - a model can have one to a finite number of copies of the same instance template model.

In Cosmo-CA, the ITM represents the CA's structure and composition in terms of its specializees. Structurally it defines the spatial dimensions of the CA and its network connectivity pattern. For specializees, it represents the structure of the specializees in a hierarchical manner. It does not have spatial location data as this is provided by the modeler when creating an instance of the CA.

Instance Model: In CoSMoS, an instance model is an instantiation of an instance template model where the multiplicity of model instances is specified In CoSMoS-CA, the instance model defines the CA's orientation, the locations of the specialized cells and the initialization values of the specialized and default cells.

Specialization:

In CoSMoS, for component-based models, specialization of a model means that the specialized now have exactly the same IO as the specializee model and has references to the state variables of the specializee model.

For cellular automata, we constrain the semantics of specialization to say that no change in network pattern (e.g., von Neumann and Moore) may occur, and no cell state variables may be added. However, the values of the cell state variables may be initialized differently. In this scenario, a CA with specialized cells means that those cells have different state variable values. We define the base cell as the underlying atomic cell from which the CAs are constructed.

Any addition/deletion in the base cell model's state variables must cascade through all specializations and all models using them.

A specializee S is defined as a set of homogeneous base cells with finite and integral dimensions. It specifies a general cell set in which all the cells must have homogeneous attributes with default values as defined by the base cell. This specializee may be used in the context of a CA by specializing it into a spatial specialization which has:

Spatial location in the context of the enclosing CA

Homogeneous attribute values for its constituent cells

A spatial specialization S_S is a specializee S that is defined in the context of an enclosing CA as being a set of homogeneous cells:

With dimensions greater than zero and less than or equal to the dimension of the enclosing CA.

Having a definite location in the CA such that the boundaries of S_s do not exceed the boundaries of the enclosing CA

Having initialization values potentially different from other cells in the CA when instantiated.

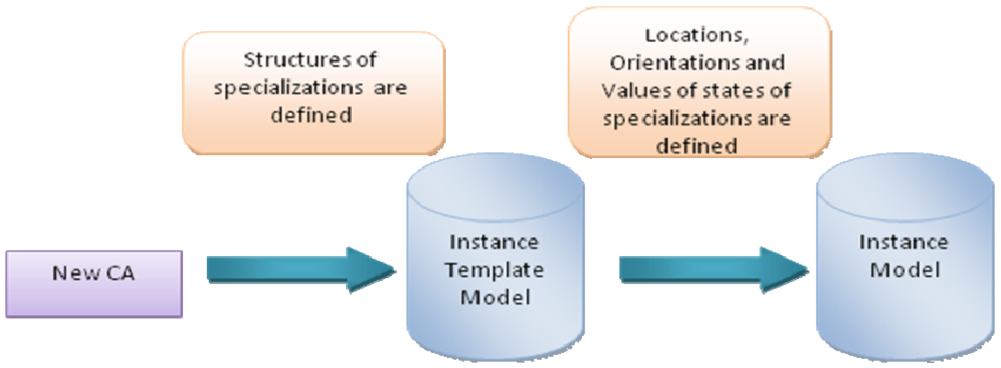

Structures of specializations are defined
Locations, Orientations and Values of states of specializations are defined
New CA
Instance Template Model
Instance Model

Figure 12. Differences between ITM and IM

Instantiation:

An individual cell cannot be instantiated independently. Only the subsuming CA can be instantiated, this would in turn instantiate the contained cells.

In other words, the cells are bound atomically to their cellspaces. A cell may not exist without a cellspace; else it becomes a non-CA component.

Instantiate from CA to cells

CHAPTER 4

DESIGN AND IMPLEMENTATION

4.1 High Level System Design

4.1.1 System Structure

The environment provides capabilities to visually create CA models and allow the modeler to manipulate the models' properties like state variables, composition etc. Some of the technical challenges that arise in the development are:

-Providing good performance even while building very large CAs

-Creating a scalable framework so that future components like support for 3D CAs or polygonal cells may be added without major code change.

-Rendering complex visualizations must be done optimally to avoid performance bottlenecks and threading issues.

-Ensuring atomicity in database operations by providing rollback mechanisms to ensure consistency.

The design for the environment reuses the client server model of CoSMo. While CoSMoS-CA reuses some existing components like the low-level database classes, the logic is implemented as a separate layer. The CoSMo Controller redirects all CA operations to the CA Controller which interacts with the above mentioned logic layer. This layer only performs high level logical decisions and delegates the lower level data requests to a CA-specific data layer which finally accesses the low-level database classes defined in CoSMo. Using this layered approach, different concerns are handled in appropriate layers and there is clear demarcation of roles. This also ensures that CoSMoS-CA can better coexist with CoSMo as they share very few hooks.

This structure is shown in the figure.

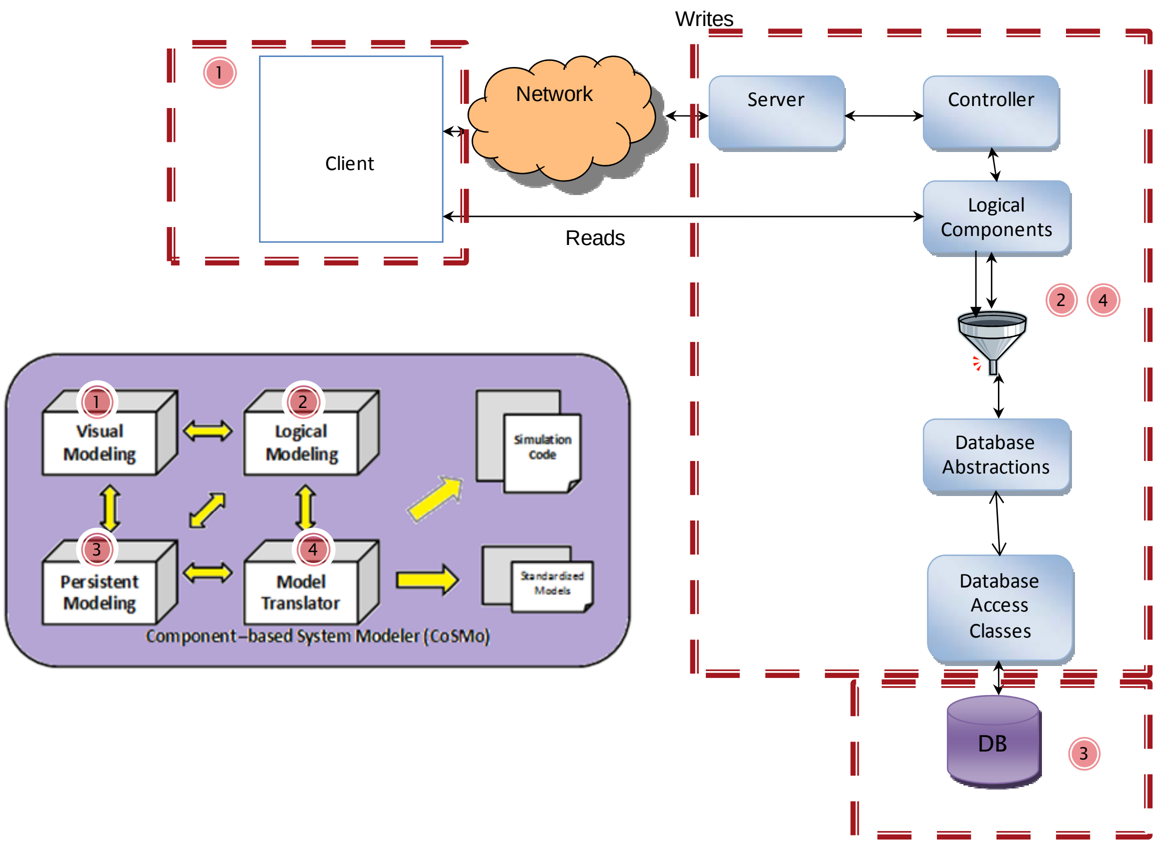


Figure 13. Conceptual Design Overview

Client: This handles all the user gestures and events from the user interface. Both CoSMoS and CoSMoS-CA share the same client.

Server: This receives all the client messages and redirects them to the respective helper classes and returns to the client the responses coming from them. For CoSMoS-CA, the server simpley redirects all requests to the CAOperationsController which performs the same task.

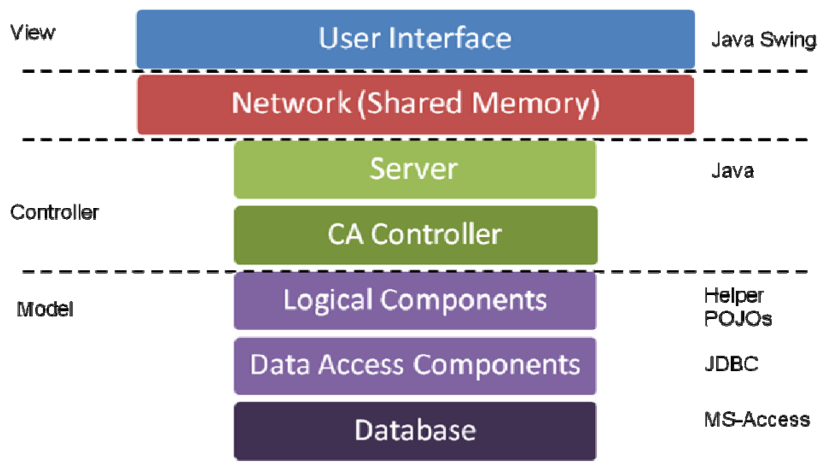


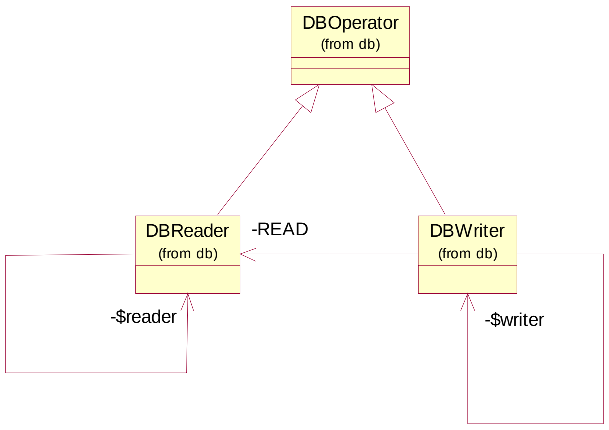
Figure 14. CoSMoS-CASystem Desgin

Logical components: These are the classes which handle the logical part of the requests. They implement the different modeling paradigms and business rules.

Database Abstractions: These are the classes which provide synchronized access to the database by treating 2 database READs or WRITEs as mutually exclusive and blocking processes.While one READ is being executed another READ has to wait until the previous READ has completed. WRITEs are also treated similarily. This is implemented by using two Singleton classes called DBReader and DBWriter.

Database Access Classes: These classes perform query processing and query preparation but leave the actual execution of the query to the underlying existing CoSMoS database classes.

The figure 16 below shows the structure of CoSMoS-CA in terms of its components. It shows the logical splitting of the functionality into 5 broad parts namely user interface, the implementation logic, database related, model export related and



utilities.

Figure 15. Singleton Classes handling DB access

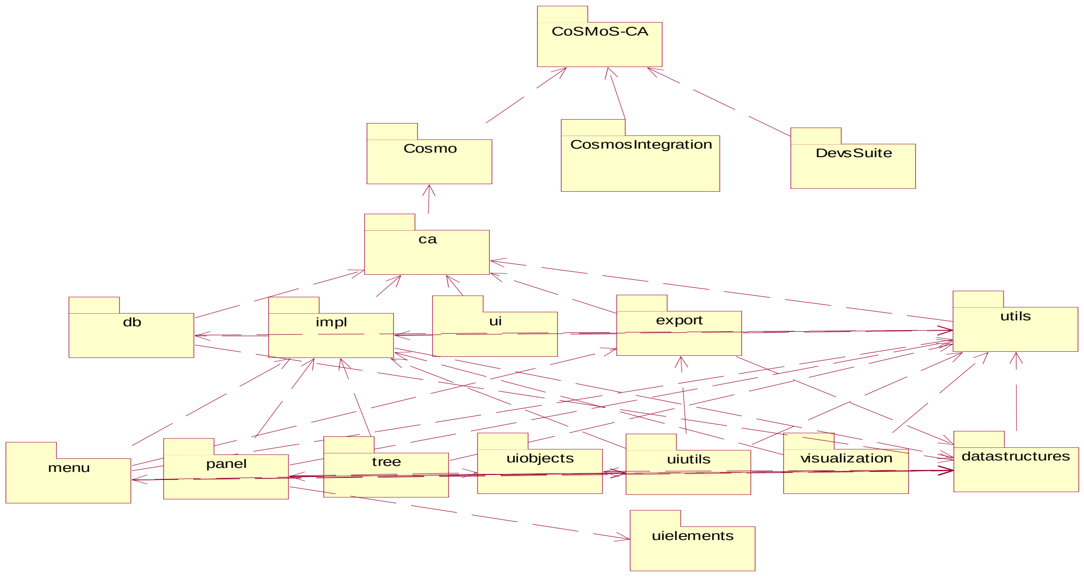


Figure 16. Component Diagram-CoSMoS-CA

4.1.2 Entity Relationship Structure

CoSMoS-CA uses 6 tables to store the structure of CAs and their state variables. These tables have no dependency on CoSMoS-non CA tables. Figure 17 below shows this structure.

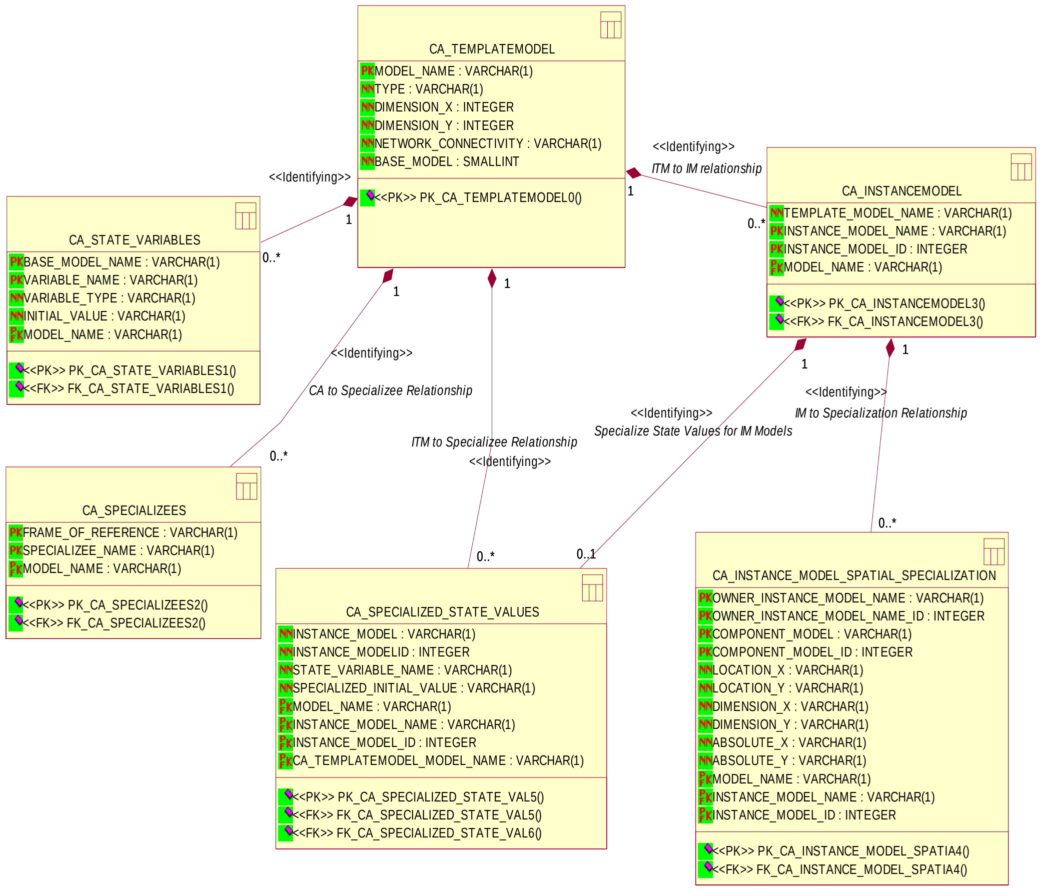

CA_TEMPLATEMODEL
PK MODEL_NAME : VARCHAR(1)
NN TYPE : VARCHAR(1)
NN DIMENSION_X : INTEGER
NN DIMENSION_Y : INTEGER
NN NETWORK_CONNECTIVITY : VARCHAR(1)
NN BASE_MODEL : SMALLINT
<<PK>> PK_CA_TEMPLATEMODEL0()
CA_STATE_VARIABLES
PK BASE_MODEL_NAME : VARCHAR(1)
PK VARIABLE_NAME : VARCHAR(1)
NN VARIABLE_TYPE : VARCHAR(1)
NN INITIAL_VALUE : VARCHAR(1)
FK MODEL_NAME : VARCHAR(1)
<<PK>> PK_CA_STATE_VARIABLES1()
<<FK>> FK_CA_STATE_VARIABLES1()
CA_INSTANCEMODEL
NN TEMPLATE_MODEL_NAME : VARCHAR(1)
PK INSTANCE_MODEL_NAME : VARCHAR(1)
PK INSTANCE_MODEL_ID : INTEGER
FK MODEL_NAME : VARCHAR(1)
<<PK>> PK_CA_INSTANCEMODEL3()
<<FK>> FK_CA_INSTANCEMODEL3()
CA_SPECIALIZEES
PK FRAME_OF_REFERENCE : VARCHAR(1)
PK SPECIALIZEE_NAME : VARCHAR(1)
FK MODEL_NAME : VARCHAR(1)
<<PK>> PK_CA_SPECIALIZEES2()
<<FK>> FK_CA_SPECIALIZEES2()
CA_SPECIALIZED_STATE_VALUES
NN INSTANCE_MODEL : VARCHAR(1)
NN INSTANCE_MODELID : INTEGER
NN STATE_VARIABLE_NAME : VARCHAR(1)
NN SPECIALIZED_INITIAL_VALUE : VARCHAR(1)
FK MODEL_NAME : VARCHAR(1)
FK INSTANCE_MODEL_NAME : VARCHAR(1)
FK INSTANCE_MODEL_ID : INTEGER
FK CA_TEMPLATEMODEL_MODEL_NAME : VARCHAR(1)
<<PK>> PK_CA_SPECIALIZED_STATE_VAL5()
<<FK>> FK_CA_SPECIALIZED_STATE_VAL5()
<<FK>> FK_CA_SPECIALIZED_STATE_VAL6()
CA_INSTANCE_MODEL_SPATIAL_SPECIALIZATION
PK OWNER_INSTANCE_MODEL_NAME : VARCHAR(1)
PK OWNER_INSTANCE_MODEL_NAME_ID : INTEGER
PK COMPONENT_MODEL : VARCHAR(1)
PK COMPONENT_MODEL_ID : INTEGER
NN LOCATION_X : VARCHAR(1)
NN LOCATION_Y : VARCHAR(1)
NN DIMENSION_X : VARCHAR(1)
NN DIMENSION_Y : VARCHAR(1)
NN ABSOLUTE_X : VARCHAR(1)
NN ABSOLUTE_Y : VARCHAR(1)
FK MODEL_NAME : VARCHAR(1)
FK INSTANCE_MODEL_NAME : VARCHAR(1)
FK INSTANCE_MODEL_ID : INTEGER
<<PK>> PK_CA_INSTANCE_MODEL_SPATIA4()
<<FK>> FK_CA_INSTANCE_MODEL_SPATIA4()
<<Identifying>>
ITM to IM relationship
<<Identifying>>
<<Identifying>>
CA to Specializee Relationship
ITM to Specializee Relationship
<<Identifying>>
<<Identifying>>
Specialize State Values for IM Models
<<Identifying>>
IM to Specialization Relationship
1
0..*
1
0..*
1
0..*
1
0..*
1
0..1
1
0..*

Figure 17. ER diagram of CoSMoS-CA

CA_TEMPLATEMODEL: This table stores the information about all instance template models like name of the model, type (cell, Ca, Specializee), dimensions, network pattern(not applicable for cell models) and the base model from which this model is composed (again not applicable for CAs).

CA_STATE_VARIABLES: Stores the state variables name,type, default value and the base cell.

CA_INSTANCEMODEL: Maps ITM models to IM models.

CA_INSTANCE_MODEL_SPATIAL_SPECIALIZATION: Maintains the FoR-Specialization relation. Describes the FoR name,the specialization name, its spatial dimension after accounting for orientation, its spatial location with respect to the FoR and the absolute location from the origin.

CA_SPECIALIZED_STATE_VALUES: Maintains the initialization values for the state variables for instance models and specializations

This table structure allows DML queries to be cascaded to all dependent tables and hence leverages the database's inherent integrity rules to minimize querying from the application and thus maintains stability of the persistence model apart from improving performance.

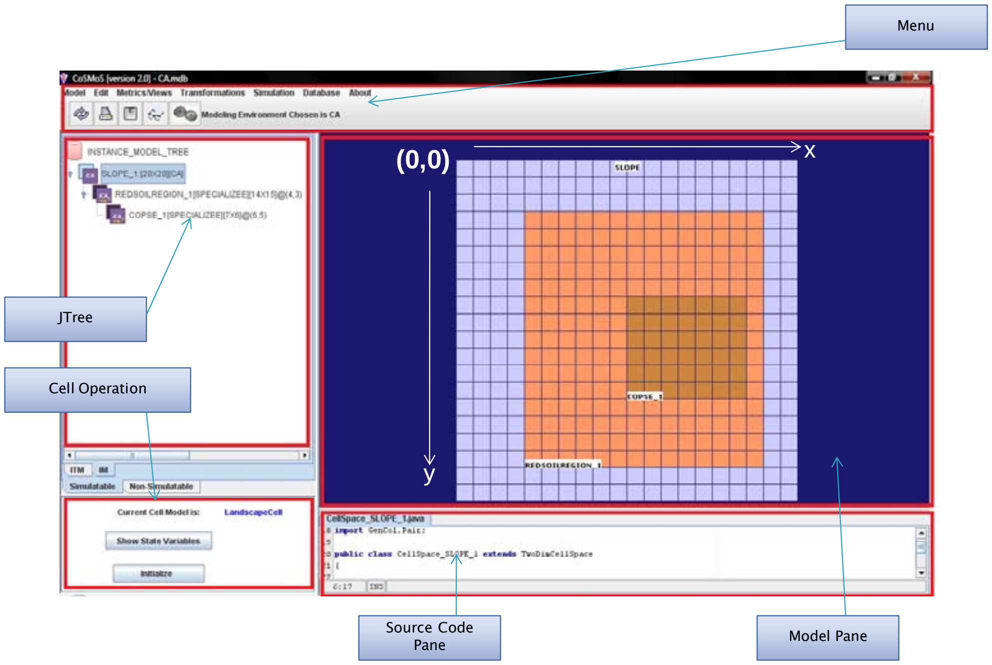
4.1.3 User Interface

The CoSMoS-CA user interface has 5 main sections:

Section	Function
Menu	Has handles to a variety of utility tasks like refreshing the JTree, launching the source code viewer, printing the current model etc.
JTree	Provides a hierarchical representation of the family of models. Also provides visual feedback to modeler about the models' name, dimensions and in case of Instance Models the coordinate within the FoR. Each node also has menus to node specific actions.
Model Pane	Provides the canvas for the model's visualization
Cell Operations Pane	Provides operations specific to the cell like view/edit the state variables and deleting the cell. All these operations will affect all CAs

	created using this cell.
Source Code Pane	Displays the exported code in the NetBeans Editor.

Table 2. Sections of CoSMoS-CA user interface

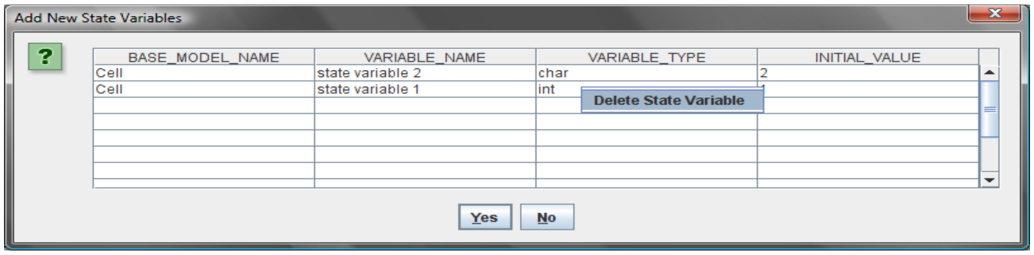

Menu
CoSMoS (version 2.0) - CA.mdb
Model Edit Metrics/Views Transformations Simulation Database About
Modeling Environment Chosen is CA
INSTANCE_MODEL_TREE
SLOPE_1 [20X20][CA]
REDSOILREGION_1[SPECIALIZEE][14X15]@(4,3)
COPSE_1[SPECIALIZEE][7X6]@(6,5)
JTree
Cell Operation
Simulatable
Non-Simulatable
Current Cell Model is: LandscapeCell
Show State Variables
Initialize
(0,0)
x
y
SLOPE
COPSE_1
REDSOILREGION_1
CellSpace_SLOPE_1.java
Source Code Pane
Model Pane

```
import GenCol.Pair;

public class CellSpace_SLOPE_1 extends TwoDimCellSpace
{
```

Figure 18. Sections of CoSMoS-CA user interface

The State Variables Grid

Another additional UI element is the grid used to view/edit the state variables. It provides a one point access to perform operations like add/delete and modify state variables.

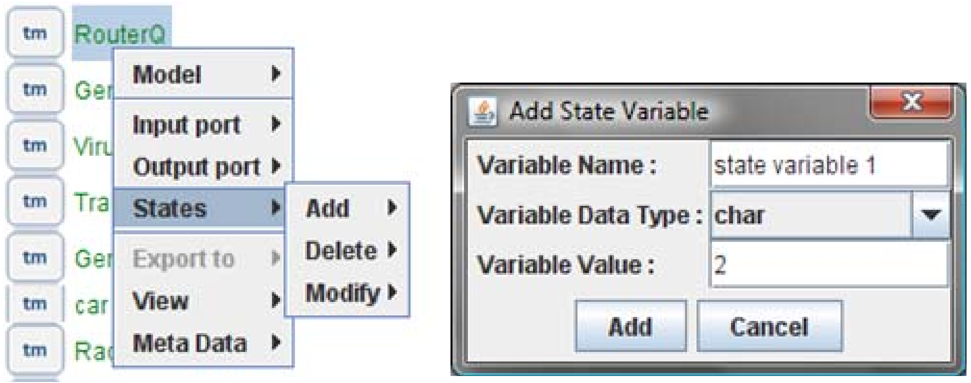


BASE_MODEL_NAME	VARIABLE_NAME	VARIABLE_TYPE	INITIAL_VALUE
Cell	state variable 2	char	2
Cell	state variable 1	int	1

Figure 19. The state variables grid

The grid extends the capabilities of a JTable by making the cells editable and by capturing user events on its cells and taking appropriate action. For example right clicking on a row pops up the delete menu using which the modeler can delete the state variable. Specifically, the delete operation affects all CAs as it deletes from the base cell.

In CoSMoS, state variables have separate dialogs for add delete and modify and do not provide a holistic view of all the variables. This grid provides this ability.



Variable Name : state variable 1
Variable Data Type : char
Variable Value : 2

Figure 20. Existing GUI elements for state variables in CoSMoS

4.2 Model Creation

CoSMoS-CA supports the idea of one model definition and multiple instantiation.

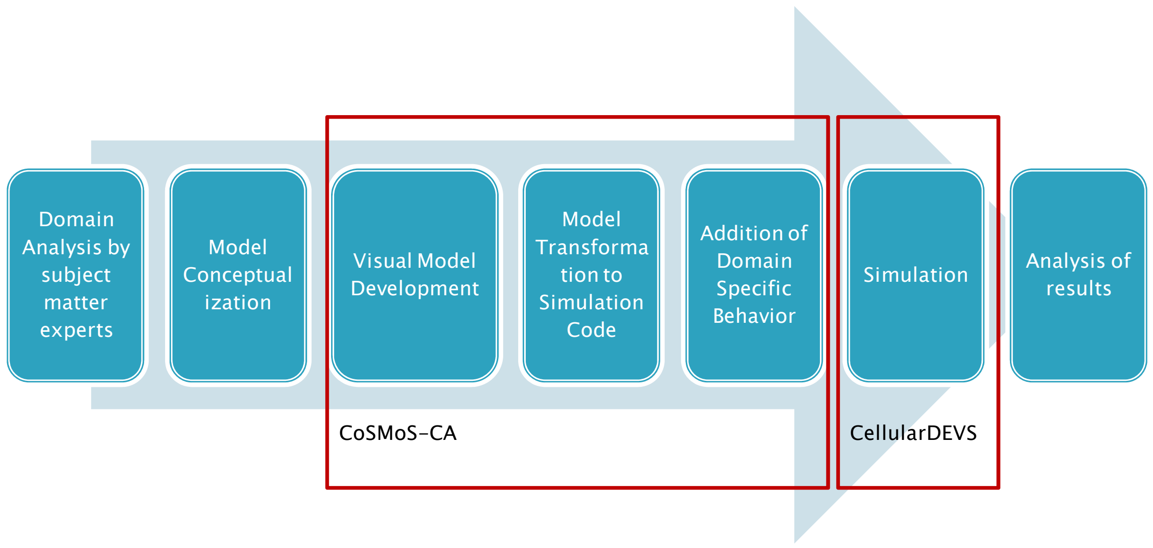

Domain Analysis by subject matter experts
Model Conceptualization
Visual Model Development
Model Transformation to Simulation Code
Addition of Domain Specific Behavior
Simulation
Analysis of results
CoSMoS-CA
CellularDEVS

Figure 21. CA Modeling Process

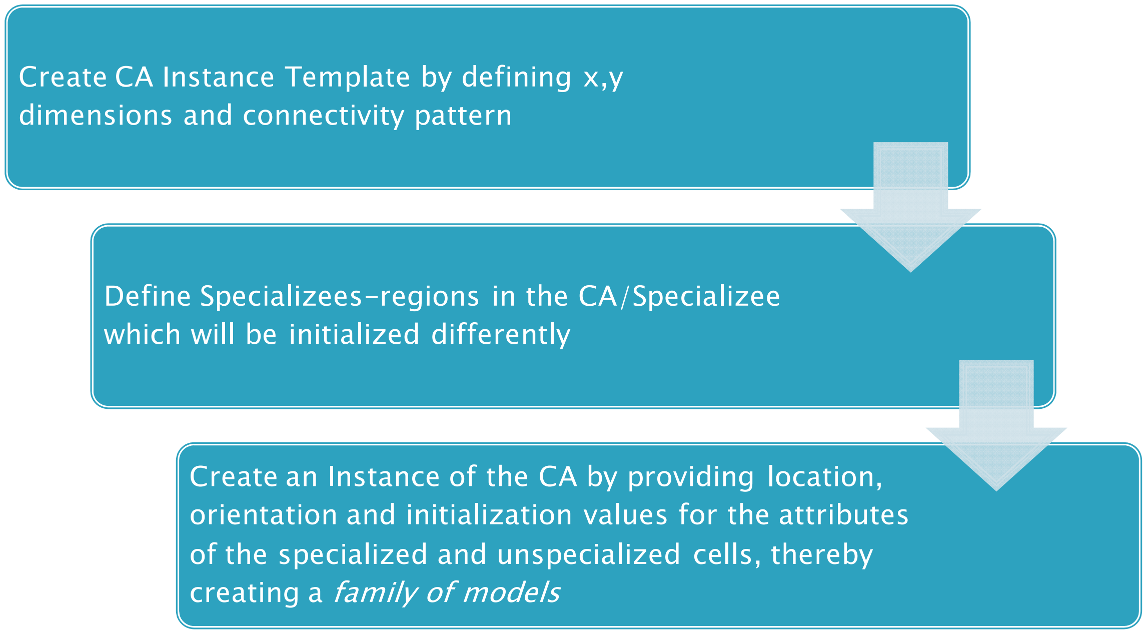

Create CA Instance Template by defining x,y dimensions and connectivity pattern
Define Specializees-regions in the CA/Specializee which will be initialized differently
Create an Instance of the CA by providing location, orientation and initialization values for the attributes of the specialized and unspecialized cells, thereby creating a *family of models*

Figure 22. Visual Model Development

In this context, the Instance Template model provides the definition of the model at an abstract level. At this level the CA is described in terms of known but value-less attributes and abstract definitions of spatial specializations which have no location or initialization. When this model is instantiated, values are assigned for the locations and orientations of specializees and for the state variables. As an analogy, the ITM represents the Class construct in Object-Orientation where the structure of the class is defined in terms of definitions of attributes and constructors. Using the same analogy, the IM represents the Object where a specific instance of the class is created with values for the attributes and constructors.

4.2.1 Instance Template Model Creation

An instance template model can be one of 3 types: a cell, a CA or a specializee. A cell represents the basic atomic unit of a CA. Before any CA can be created, the base cell needs to be defined in terms of its name, shape and state variables. Currently CoSMoS supports only square cells. Once a cell has been created a family of CAs and their specializees may be created. Since currently the specification of CoSMoS-CA does not extend to composable CAs, where CAs created by different cells may, the family of CAs created using one cell model have no connection to CAs created using another cell model. Hence when CoSMoS-CA is launched the modeler has to choose which family of models to load, by selecting the cell model from the list of all cell models see Figure 21.

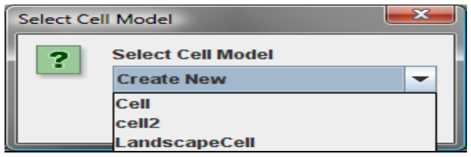
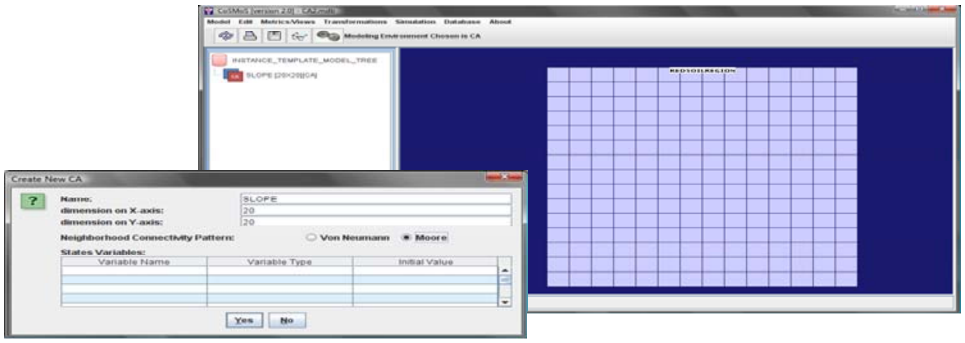


Figure 23. Selecting a Cell model

4.2.1.1 Creating a CA

4.2.1.1.1 Algorithm

Creating a CA requires that the modeler knows which cell model he wants to create the CA out of. Once the cell model has been selected the modeler needs to define the CA's unspecialized structure by specifying the name of the CA, its spatial dimensions and network connectivity pattern. See figure 22 which displays the form which prompts the user to provide this information. Currently, CoSMoS-CA supports only Von Neumann and Moore connectivity patterns. Once the modeler provides the information the CA is created in the database using the following simple algorithm.


CoSMoS [version 2.0] - CA2.mdl
Model Edit Metrics/Views Transformations Simulation Database About
Modeling Environment Chosen is CA
INSTANCE_TEMPLATE_MODEL_TREE
SLOPE [20×20][CA]
REDSOILREGION
Create New CA
Yes
No

Name: SLOPE
dimension on X-axis: 20
dimension on Y-axis: 20
Neighborhood Connectivity Pattern: Von Neumann [✓ Moore]
States Variables:

Variable Name	Variable Type	Initial Value

Figure 24. Creating a new CA

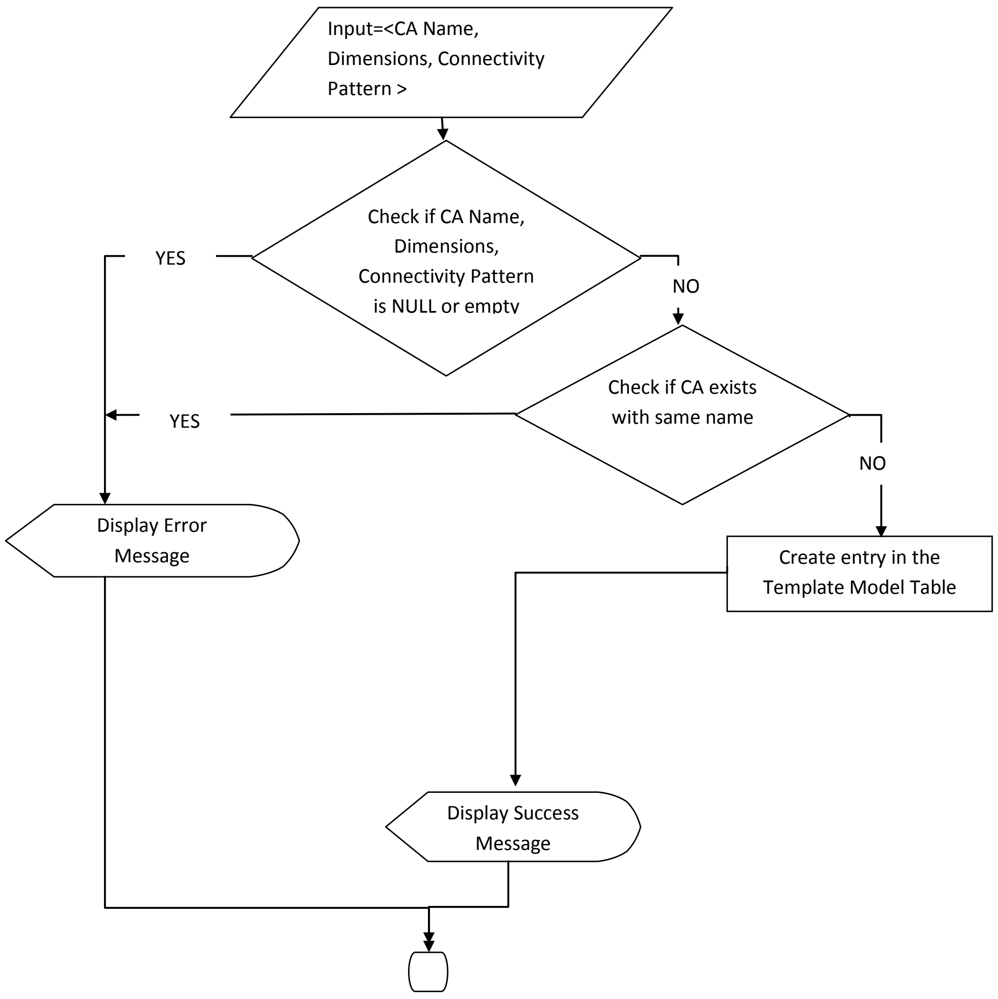

Input=<CA Name, Dimensions, Connectivity Pattern >
Check if CA Name, Dimensions, Connectivity Pattern is NULL or empty
YES
NO
Check if CA exists with same name
YES
NO
Display Error Message
Create entry in the Template Model Table
Display Success Message

Figure 25. Creating a CA

4.2.1.1.2 Class Diagram

The class diagram in Figure 24 shows the classes responsible for getting the CA's structural data from the user. SesmGUI is the class that represents the CoSMoS-CA user interface and handles all user actions. CAPanels holds the logic for generating and displaying the form and doing basic input data validations. Once it receives the validated user interface data an event is created. sesmEevent represents this event. This event is placed on the CoSMoS network and sent to the server.

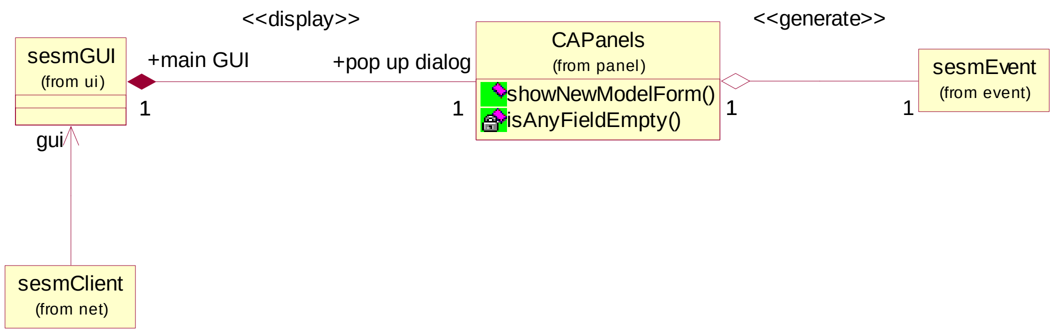


Figure 26. Class Diagram- UI part of creating a new CA

Once the event is received by the server, it is redirected to the CAOperationsController which redirects it to the CA related classes and the helper methods. In this case, the event is sent to the CAAdd class which implements the above algorithm. It does this using supporting classes like CAQuery and the lower level DB related classes which interface with the existing CoSMoS database components.

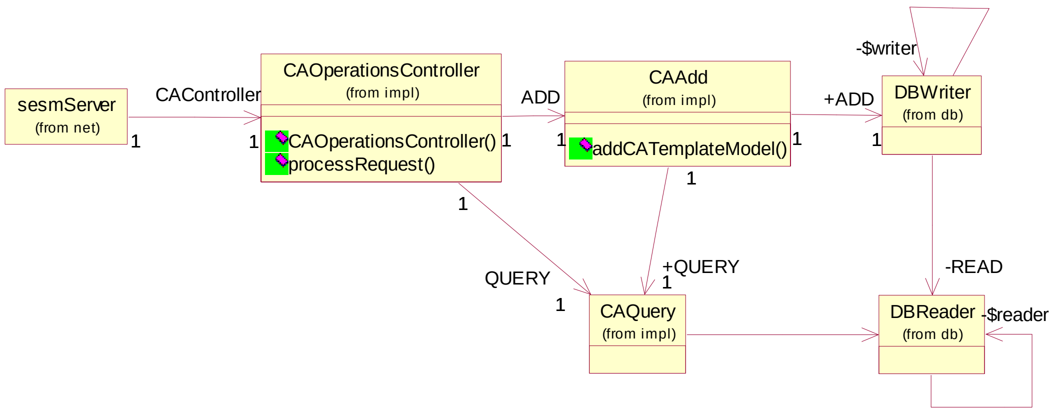


Figure 27. Class Diagram-Server components for creating a new CA

4.2.1.1.3 Sequence Diagram

The figure 26 shows the sequence diagram for interaction between the user interface and the modeler when creating a new CA. When the modeler invokes the "Create new CA" operation the following methods get executed:

showNewModelForm: This method creates the GUI form to allow the modeler to specify CA structure data. The integer argument is to identify which form to create as the method is reused in other components. This method returns on Object array which has the above mentioned data.

send: This method puts the sesmEvent on the shared memory channel.

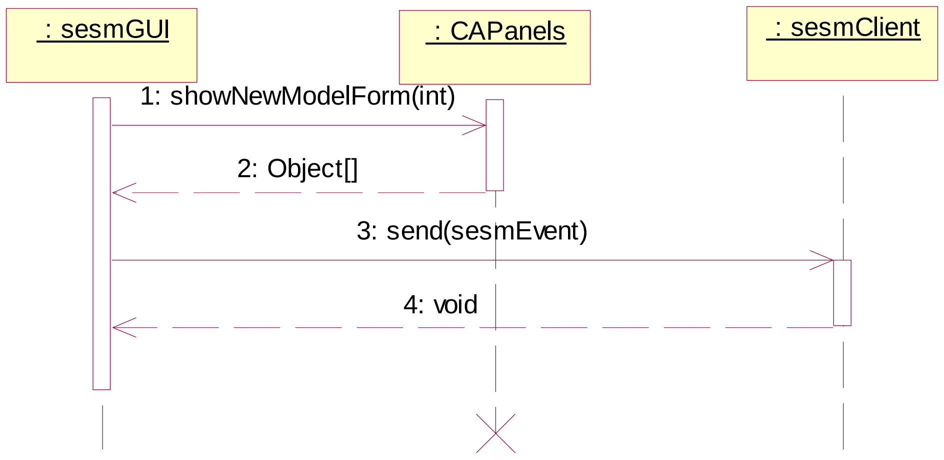


Figure 28. Sequence Diagram-UI part of creating a new CA

The sequence diagram in figure 27 shows the steps for the server part of creating a new CA. Some of the key methods are:

isNameExists: This method checks if the model name is unique in the database as, in order to maintain consistency in the database all model names have to be unique. The argument is the model name.

createNewCA: This method inserts a record into the CA_TEMPLATEMODEL table. The arguments are the CA name, the type ("CA" /"Cell" /"Specializee"), the dimensions, the connectivity pattern ("Moore" /"Von Neumann") and the base model

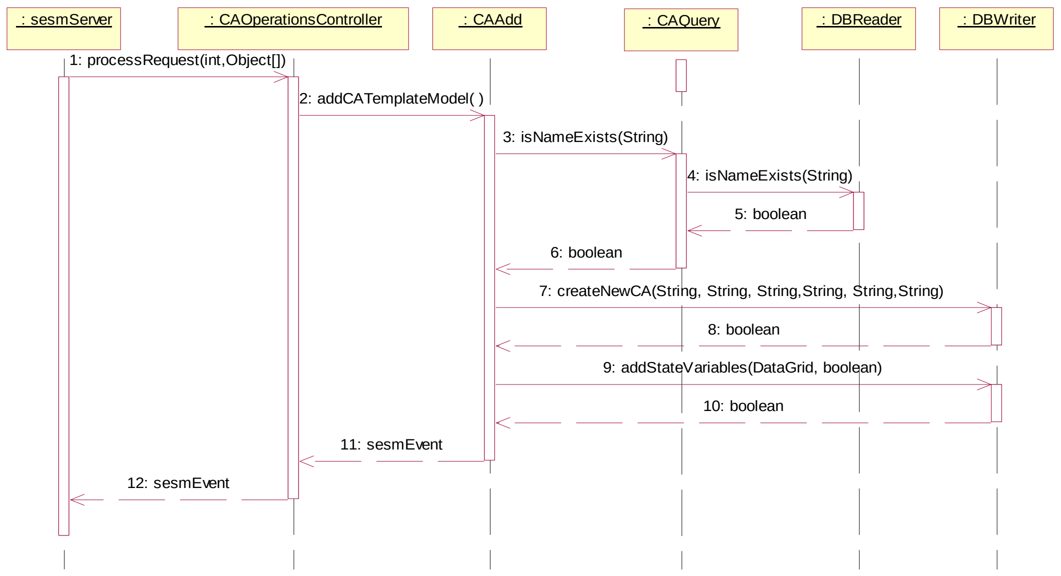


Figure 29. Sequence Diagram-Server part of creating a new CA

4.2.1.2 Creating Specializees

Specializees are sets of cells in a given CA which have different initialization values. These sets have definite spatial dimension but do not have a location specified or values for its state variables defined till instantiation.

4.2.1.2.1 Algorithm

The algorithm to creating a specializee is similar to the one used to create a CA, since even by theory, a specializee is a set of homogeneous cells, in other words a CA. The only check made is to see if the specializee has spatial dimensions which would allow it to be within the boundaries of the context CA or specializee.The subsequent figure shows the logic in validating whether the specializee is within the spatial bounds of the frame of reference.

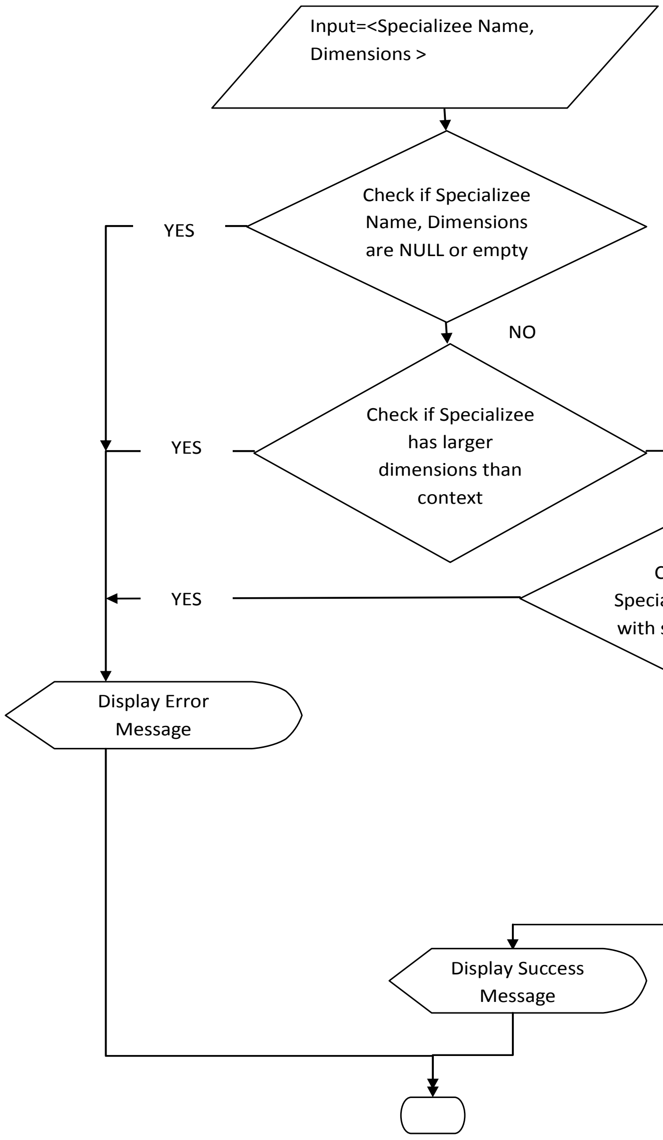
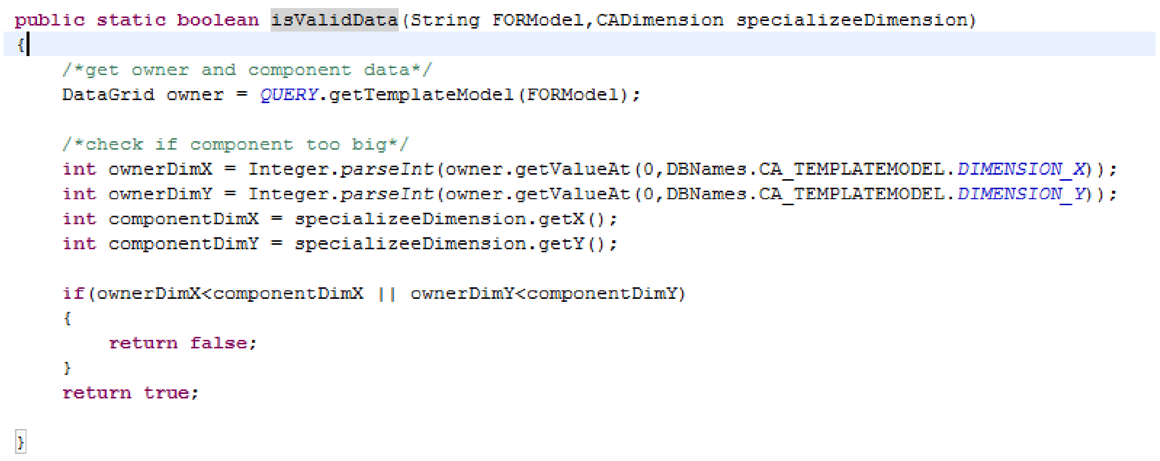

Input=<Specializee Name, Dimensions >
Check if Specializee Name, Dimensions are NULL or empty
YES
NO
Check if Specializee has larger dimensions than context
YES
NO
Check if Specializee exists with same name
YES
NO
Display Error Message
Create entry in the Template Model Table
Create entry in the Specializee Table
Display Success Message

Figure 30. Algorithm for creating a new Specializee



```
public static boolean isValidData(String FORModel,CADimension specializeeDimension)
{
    /*get owner and component data*/
    DataGrid owner = QUERY.getTemplateModel(FORModel);

    /*check if component too big*/
    int ownerDimX = Integer.parseInt(owner.getValueAt(0,DBNames.CA_TEMPLATEMODEL.DIMENSION_X));
    int ownerDimY = Integer.parseInt(owner.getValueAt(0,DBNames.CA_TEMPLATEMODEL.DIMENSION_Y));
    int componentDimX = specializeeDimension.getX();
    int componentDimY = specializeeDimension.getY();

    if(ownerDimX<componentDimX || ownerDimY<componentDimY)
    {
        return false;
    }
    return true;

}
```

Listing 1. Checking spatial validity of the specializee against Frame of Reference(FOR)

4.2.1.2.2 Class Diagram

The class diagram in figure 29 shows some of the classes used at the client side user interface.

CATemplateModelMenu: This class is a UI component that creates all the menu options when a user clicks on the model on the JTree.

UIManager: This class performs UI related controller role. All user events are sent to the UIManager which redirects to the corresponding helper classes to service them.

AddSpecializeePanel: holds the logic for generating and displaying the form and doing basic input data validations. Once it receives the validated user interface data an event is created. sesmEvent represents this event. This event is placed on the CoSMoS network and sent to the server.UIUtils:Contains the validation methods and other utility methods specific to the UI.

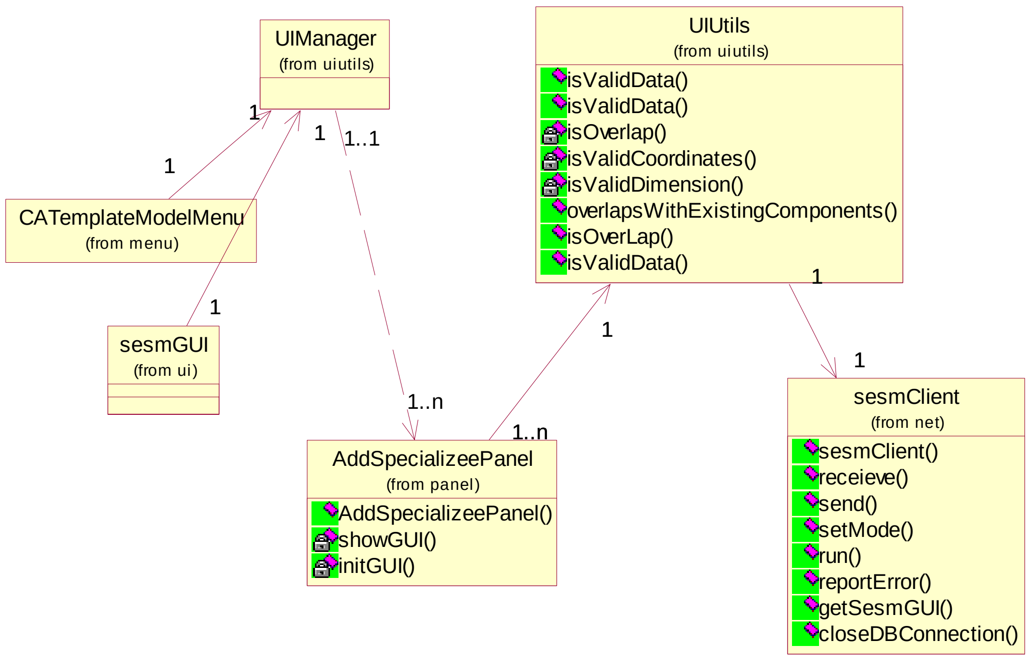


Figure 31. Class Diagram- Client side components for creating a new Specializee

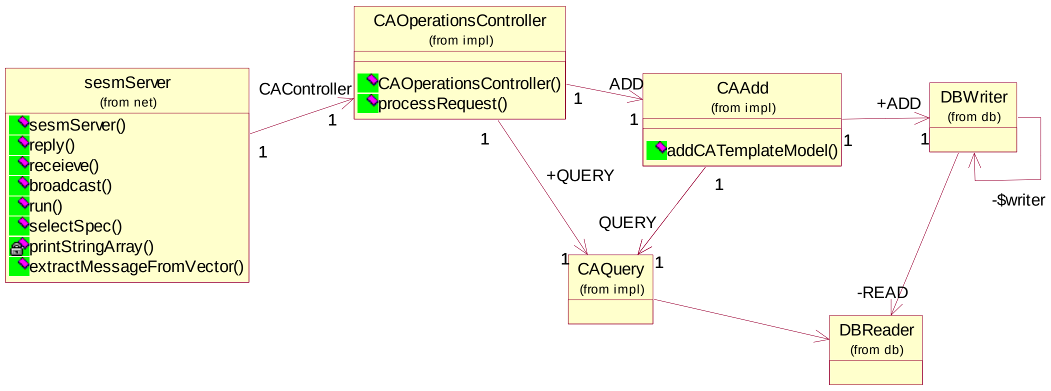


Figure 32. Class Diagram- Server side components for creating a new Specializee

4.2.1.2.3 Sequence Diagram

The figure 31 shows the sequence diagram for creating a new specializee at the server.

It is almost similar to the creation of a CA except for an additional step where the CA_SPECIALIZEES table is updated with a record to show FOR-Specializee relationship. Again the same components are reused.

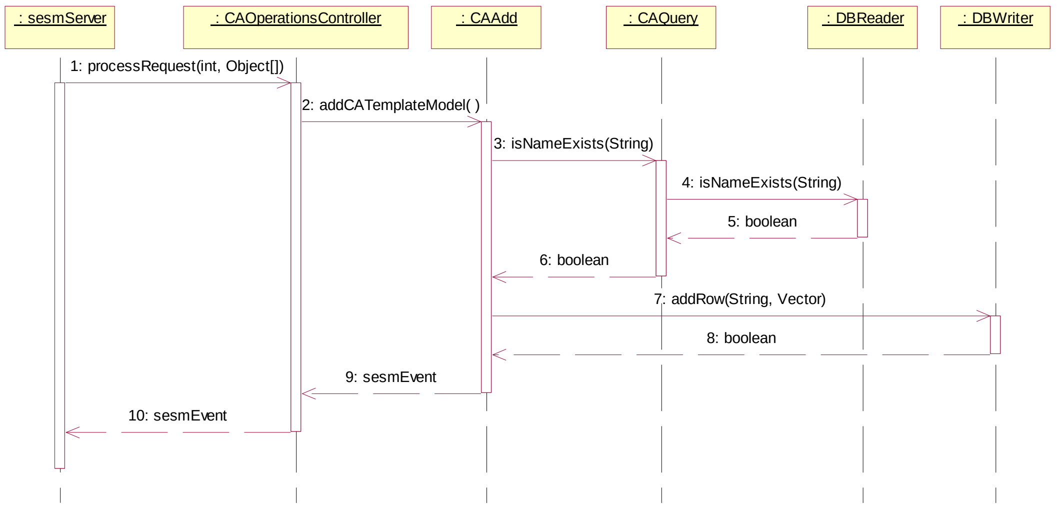


Figure 33. Sequence Diagram- Creating a specializee

4.2.2 Creating Instance Models

Instance Models represent a specification of the ITM but with values for CA's state variables, location and orientation for the specializees and values for the specialized cells.

4.2.2.1 Algorithms

The algorithm in figure 32 shows the logic involved in creating an instance model of a CA.

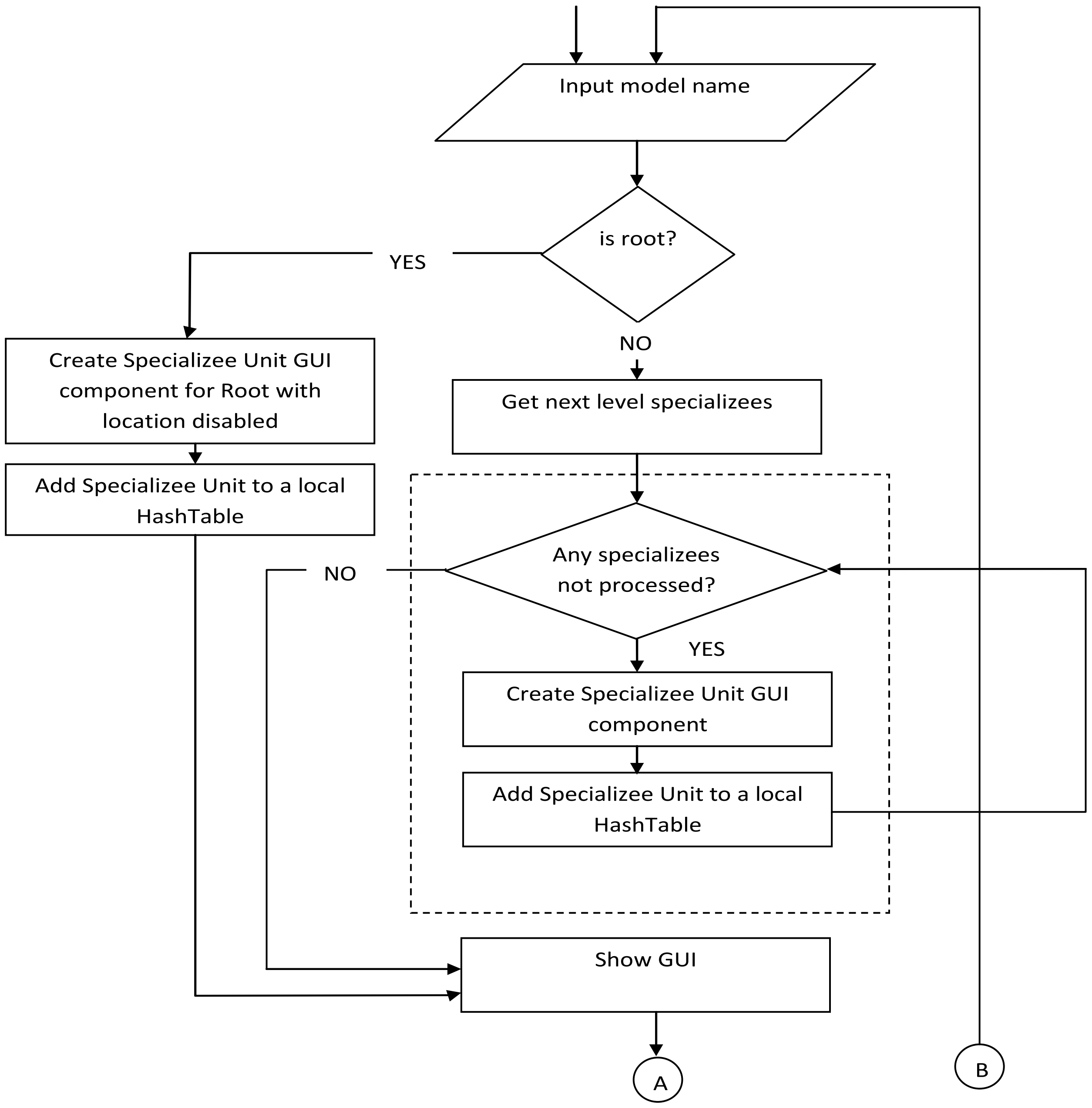

Input model name
is root?
YES
NO
Create Specializee Unit GUI component for Root with location disabled
Add Specializee Unit to a local HashTable
Get next level specializees
Any specializees not processed?
NO
YES
Create Specializee Unit GUI component
Add Specializee Unit to a local HashTable
Show GUI
A
B

Figure 34. Algorithm- Creating Instance Model Client Side

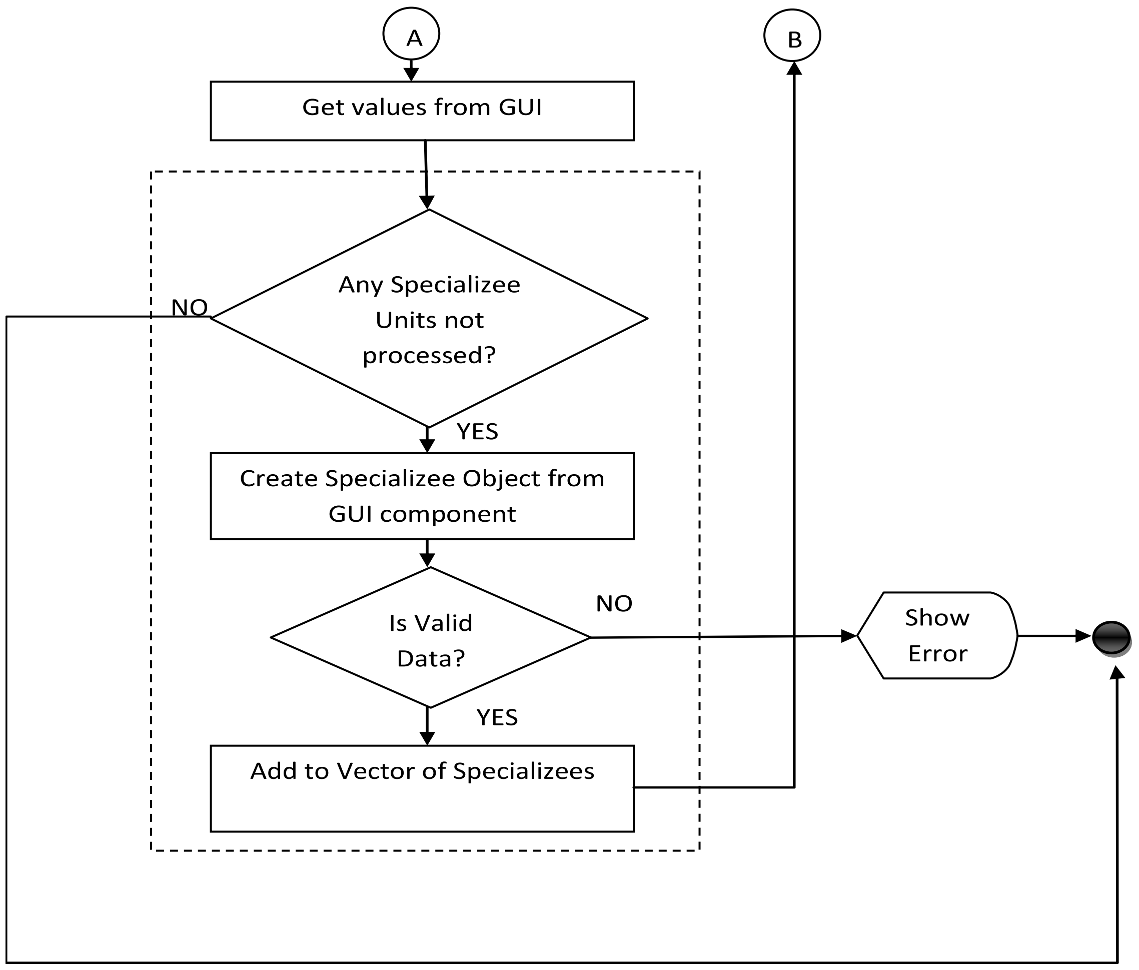
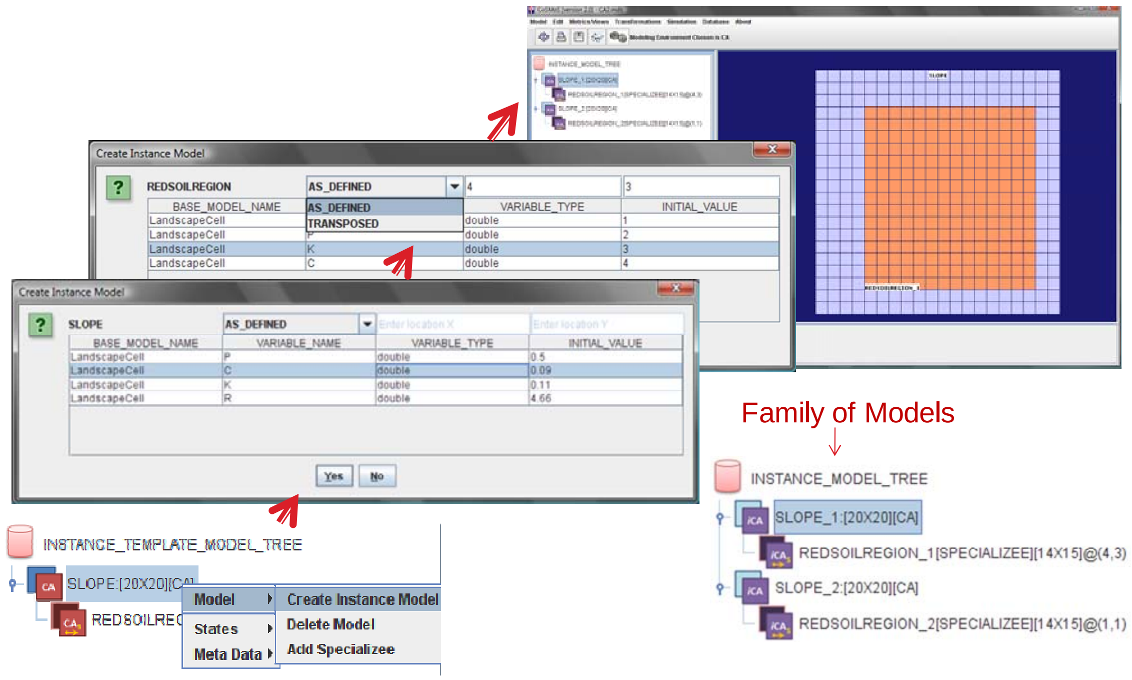

A
B
Get values from GUI
Any Specializee Units not processed?
NO
YES
Create Specializee Object from GUI component
Is Valid Data?
NO
YES
Show Error
Add to Vector of Specializees

Figure 35. Algorithm- Creating Instance Model Client Side

When the modeler selects a CA to be instantiated, he needs to specify the orientation, the location of the model within its frame of reference (FoR) and the initialization of the state variables. This specification needs to be made for each and every specializee and for the unspecialized regions of the CA, the only exception being that for the CA the location fields are disabled as the CA has no FoR. After taking in values for the CA, which is the root of the model tree, a representation of which is shown in the JTree, the modeler has to provide instantiation values for the first level specializees dialog appears as shown in figure 33.



Create Instance Model

REDSOILREGION AS_DEFINED 4 3

AS_DEFINED
TRANSPOSED

BASE_MODEL_NAME		VARIABLE_TYPE	INITIAL_VALUE
LandscapeCell		double	1
LandscapeCell	P	double	2
LandscapeCell	K	double	3
LandscapeCell	C	double	4

Create Instance Model

SLOPE AS_DEFINED Enter location X Enter location Y

BASE_MODEL_NAME	VARIABLE_NAME	VARIABLE_TYPE	INITIAL_VALUE
LandscapeCell	P	double	0.5
LandscapeCell	C	double	0.09
LandscapeCell	K	double	0.11
LandscapeCell	R	double	4.66

Figure 36. Specifying Specializees while creating Instance models

After this a Depth First traversal is followed to process the sub specializees using recursion. The algorithm shows how the model tree is traversed and allows the user to fully define each specializee. At the end of the algorithm, the final vector contains all the Specializee objects and is sent to the server.

At the server, 3 tables have to be updated in a particular sequence described below in order to preserve integrity constraints in the database. First an entry must be made in the CA_INSTANCEMODEL , this maintains the relation between ITM and the IM models. If this succeeds, then an entry is made in the table CA_INSTANCE_MODEL_SPATIAL_SPECIALIZATION which stores all the spatial data for the specializee and also stores the FoR-specializee relationship. If this INSERT succeeds the CA_SPECIALIZED_STATE_VALUES table is updated the specializee-specific initialization values for the specializee. At any point if any INSERT fails, the top most entry at the CA_INSTANCEMODEL table is deleted as this would cascade to all related tables.

Every Instance Model a unique ID is assigned to uniquely identify it from other instances. This unique ID implemented as an incrementing integer which is incremented for each family of instance models. So when a new IM is created, its id is the MAX(ID) from the database +1. When an IM is deleted the IDs are rearranged accordingly.

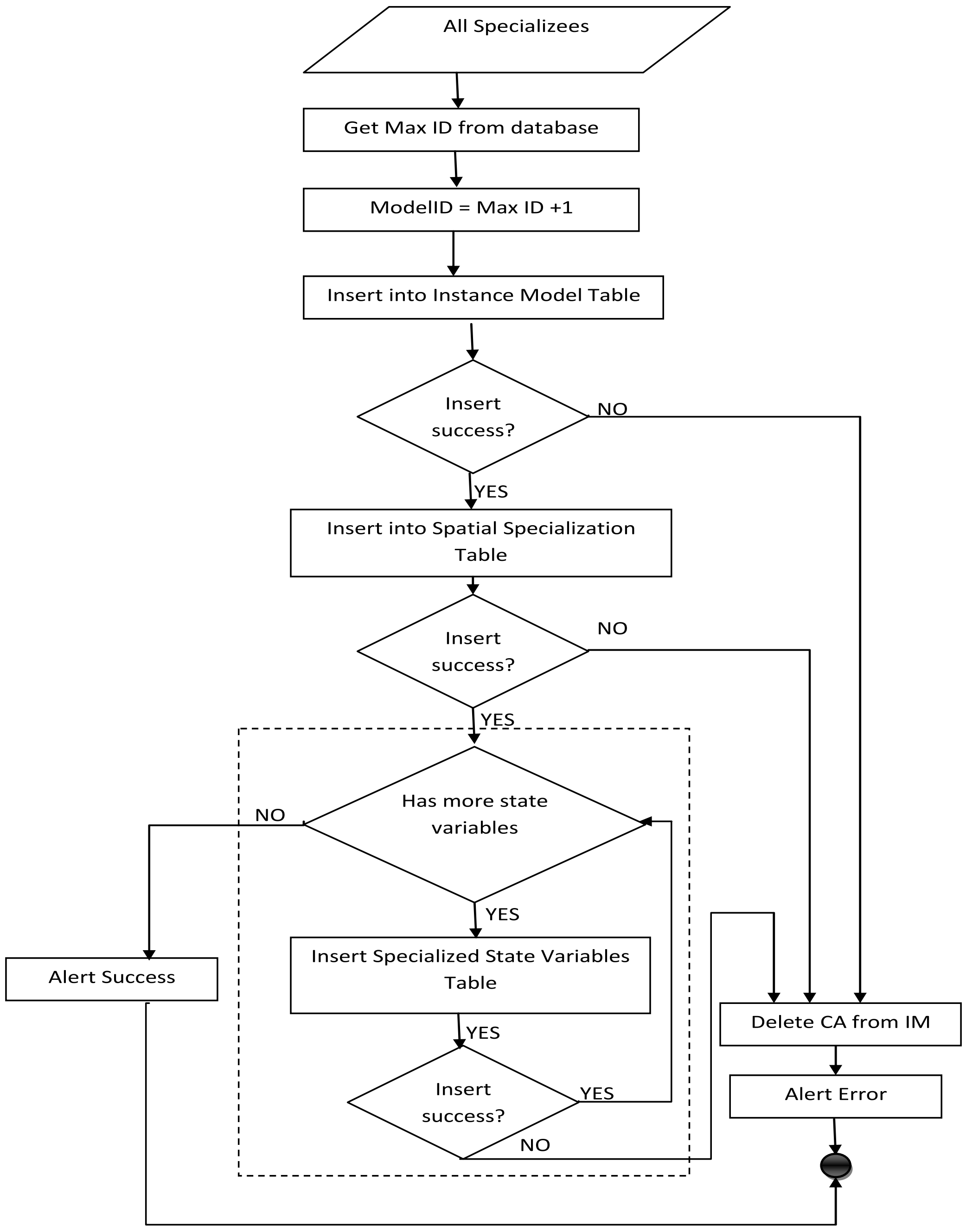


Figure 37. Algorithm- Creating Instance Model Server Side

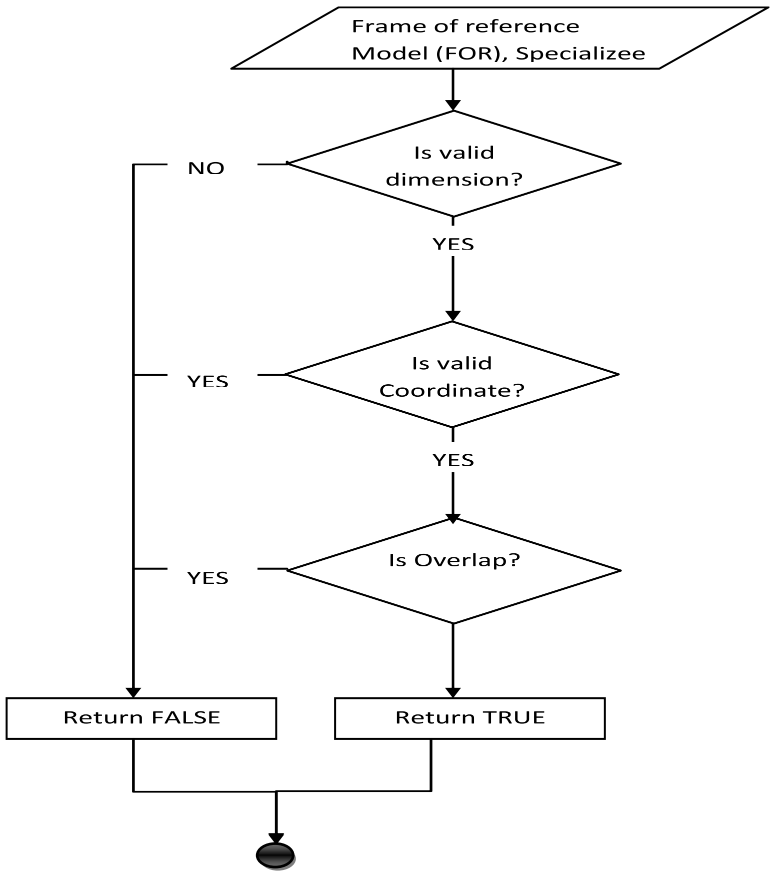


Figure 38. Algorithm- Validating IM spatial data

For every specializee that is processed, the spatial attributes like dimensions and coordinates are validated with the rest of the model. In particular, 3 checks are made:

IsValidDimension: this checks if, after taking into account the orientation, the specializee will fit within the FoR model. The process exits if it does not.

IsValidCoordinates: this checks if after taking into account the orientation, the specializee will be within the spatial bounds of the FoR. The difference between this check and the previous one is that isValidDimension only checks for size fit and does not take into account where it is placed in the FoR. The isValidCoordinates check looks at the actual location and then

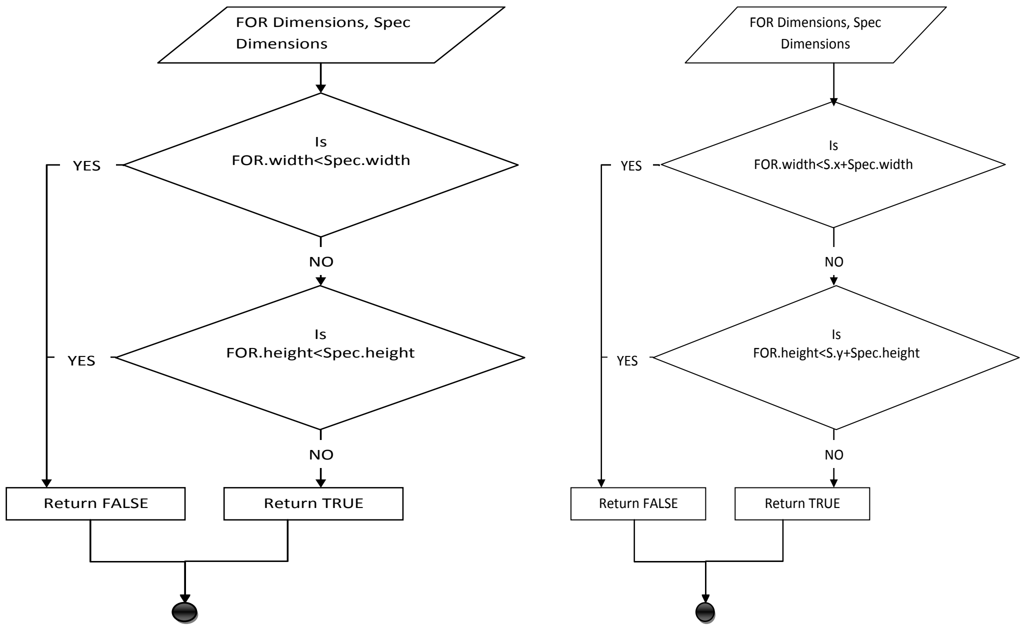

FOR Dimensions, Spec Dimensions
Is FOR.width<Spec.width
YES
NO
Is FOR.height<Spec.height
YES
NO
Return FALSE
Return TRUE
FOR Dimensions, Spec Dimensions
Is FOR.width<S.x+Spec.width
YES
NO
Is FOR.height<S.y+Spec.height
YES
NO
Return FALSE
Return TRUE

Figure 39. Algorithm- Validating IM spatial data- Dimensions and Coordinates

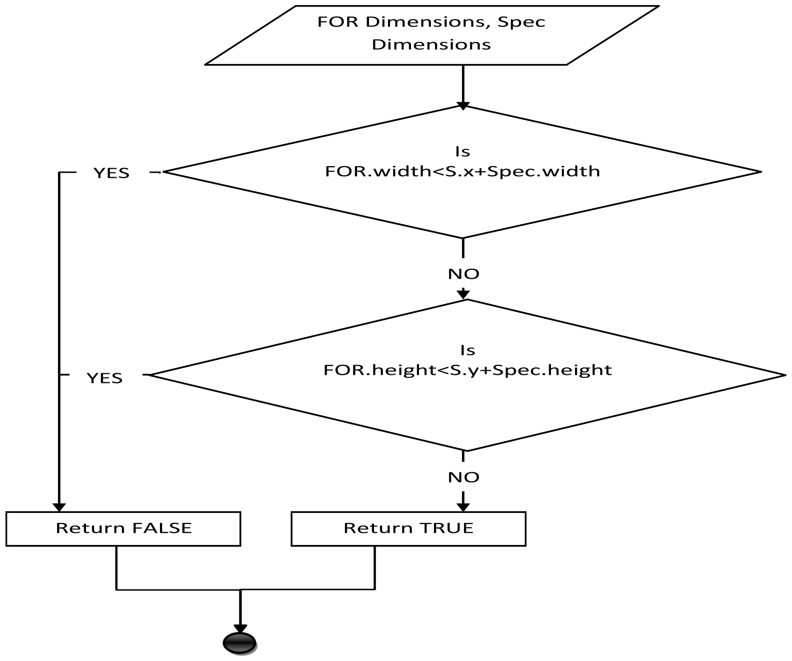

FOR Dimensions, Spec Dimensions
Is FOR.width<S.x+Spec.width
YES
NO
Is FOR.height<S.y+Spec.height
YES
NO
Return FALSE
Return TRUE

Figure 40. Algorithm- Validating IM spatial location to check for overlaps with other specializees

calculates if, placing the specializee at that point would cause it to exceed the bounds of the FoR. The isOverlap takes all the first level specializees for the FoR and checks if the current model,if placed at the coordinates specified, would overlap an existing specializee. If so, it alerts the user and exits. The above algorithms show the following logical processes:

4.2.2.2 Class Diagram

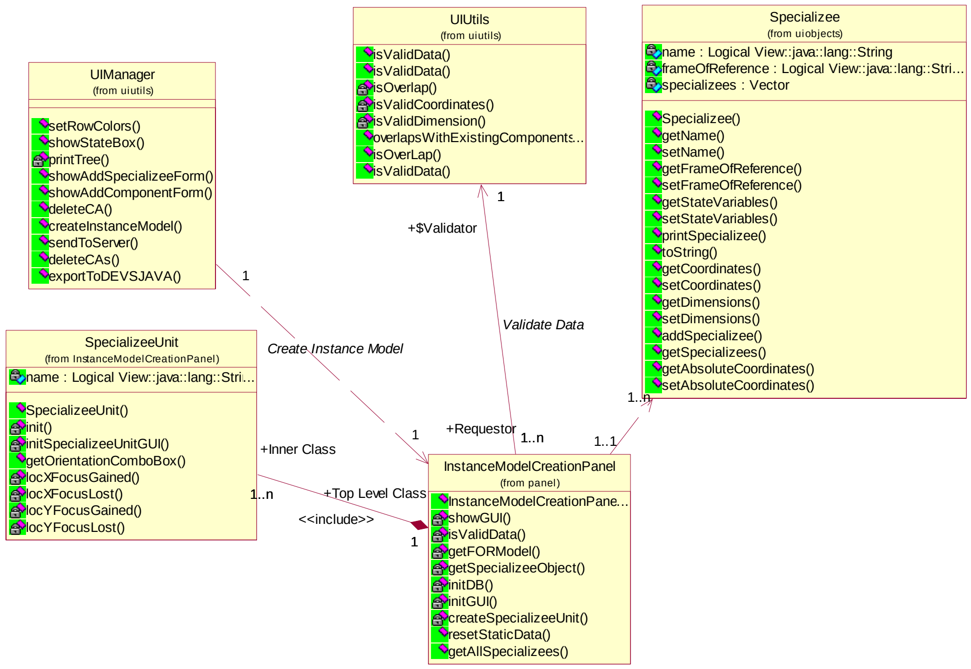


Figure 41. Class Diagram-Classes for Instance Model Creation

The class diagram in figure 38 shows some of the key classes in the IM creation process. The InstanceModelCreationPanel class contains the GUI related logic and also encapsulates a nested SpecializeeUnit class that provides an Object wrapping of the GUI element. UIUtils contains all the methods for validating the spatial data and implements the validation algorithms discussed above. The Specializee class is a one-on-one mapping with the logical specification of specializee as defined in the theoretical specification and in the

CA_INSTANCE_MODEL_SPATIAL_SPECIALIZATION table. This object is created from the SpecializeeUnit defined above.

4.2.2.3 Sequence Diagram

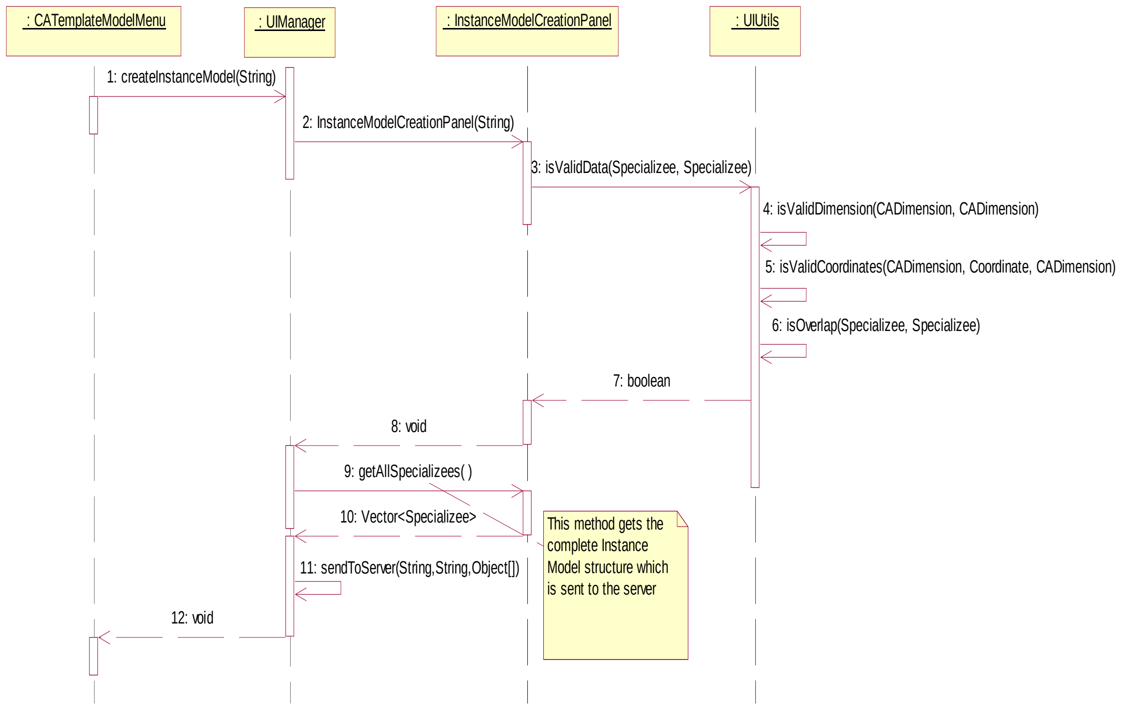

: CATemplateModelMenu
: UIManager
: InstanceModelCreationPanel
: UIUtils
1: createInstanceModel(String)
2: InstanceModelCreationPanel(String)
3: isValidData(Specializee, Specializee)
4: isValidDimension(CADimension, CADimension)
5: isValidCoordinates(CADimension, Coordinate, CADimension)
6: isOverlap(Specializee, Specializee)
7: boolean
8: void
9: getAllSpecializees( )
10: Vector<Specializee>
11: sendToServer(String,String,Object[])
12: void
This method gets the complete Instance Model structure which is sent to the server

Figure 42. Sequence Diagram- Instance Model Creation

The above sequence diagram describes the sequences of interactions between the components during the IM creation. This describes the interactions involved in creating the forms for specifying the specializee using the Depth First traversal algorithms described above. After the algorithm is executed the method getAllSpecializees () is used to retrieve the vector containing the Specializees which is then sent to the server to be inserted into the database.

4.2.3 Model Visualization

4.2.3.1 Algorithm

The aim of the visualization is to provide a clear display mechanism which the modeler could use as a visual aid to help him create complex models. Some of the challenges encountered were to be able to provide a high-performance UI even while rendering very large CAs with highly complex specialized regions. Also, to make use of the available display space and present the models in a consistent and efficient manner. The CA is visualized in a set rectangular region called the Real Estate. The high-level algorithm in figure 40 describes the process.

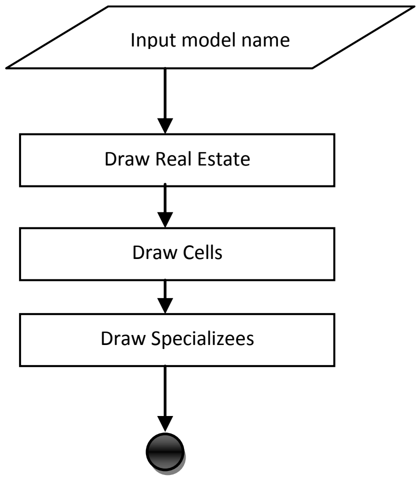


Figure 43. Algorithm- Visualization of Instance Model

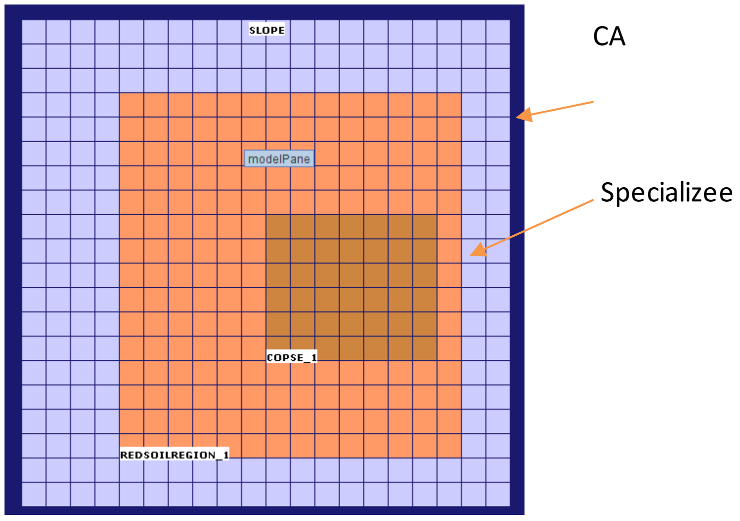


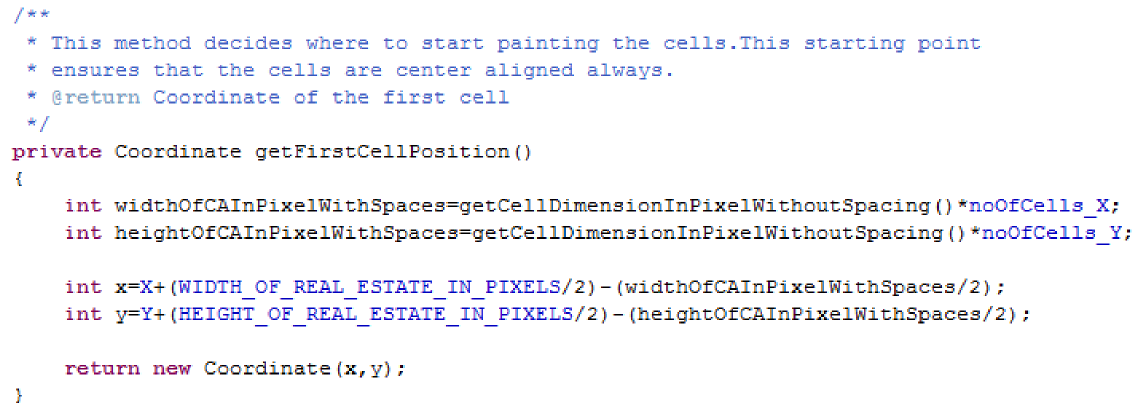
Figure 44. CA Visualization Components

The algorithm first draws the Real Estate which is a rectangular canvas on which all the visualizations are made. The size of the cell is calculated using the following formula as the CA needs to make maximum use of the real estate. The cell spacing a 1 pixel gap between two adjacent cells, the noOfCells_X and noOfCells_Y refer to the number of cells along the X-axis and the number of cells on the Y-axis. This returns the cell dimension in pixels.

```
/**
 * This method gets the pixel size of each cell.
 * if the no of cells is greater than the available real estate then we return
 * zero and a flat rectangle is shown.
 * Else,the min(length,breadth) is chosen where,
 *         length= no of pixels in real estate's width/no  of cells on x-axis
 *         breadth=no of pixels in real estate's height/no of cells on y-axis
 * @return dimension of each cell in pixels
 */
private int getCellDimensionInPixel()
{
    return Math.min(
            (WIDTH_OF_REAL_ESTATE_IN_PIXELS-(this.noOfCells_X*CELLSPACING))/this.noOfCells_X,
            (HEIGHT_OF_REAL_ESTATE_IN_PIXELS-(this.noOfCells_Y*CELLSPACING))/this.noOfCells_Y);
}
```

Listing 2. Calculating the cell dimension

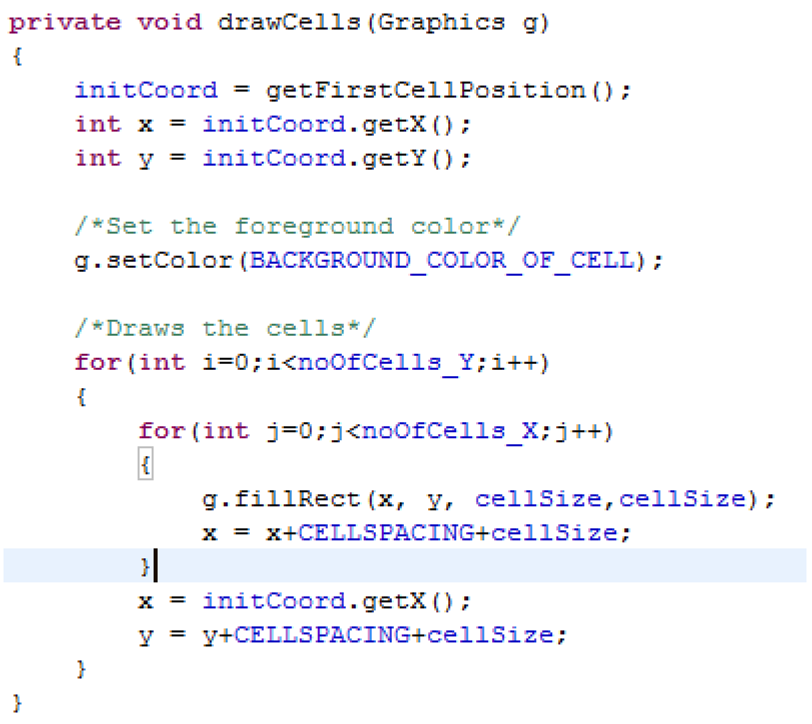
Once the cell dimension is determined, the starting point of the CA needs to be calculated so as to always place it in the middle of the real estate. Using this cell dimension ensures that the real estate will be used up at least on one axis.



```
/**
 * This method decides where to start painting the cells.This starting point
 * ensures that the cells are center aligned always.
 * @return Coordinate of the first cell
 */
private Coordinate getFirstCellPosition()
{
    int widthOfCAInPixelWithSpaces=getCellDimensionInPixelWithoutSpacing()*noOfCells_X;
    int heightOfCAInPixelWithSpaces=getCellDimensionInPixelWithoutSpacing()*noOfCells_Y;

    int x=X+(WIDTH_OF_REAL_ESTATE_IN_PIXELS/2)-(widthOfCAInPixelWithSpaces/2);
    int y=Y+(HEIGHT_OF_REAL_ESTATE_IN_PIXELS/2)-(heightOfCAInPixelWithSpaces/2);

    return new Coordinate(x,y);
}
```

Listing 3. Calculating the position of the top left most cell

Once this is calculated the Cells are rendered using a simple nested loop function which has complexity of O(xy) where x and y are the number of cells on x and y axes respectively.



```
private void drawCells(Graphics g)
{
    initCoord = getFirstCellPosition();
    int x = initCoord.getX();
    int y = initCoord.getY();

    /*Set the foreground color*/
    g.setColor(BACKGROUND_COLOR_OF_CELL);

    /*Draws the cells*/
    for(int i=0;i<noOfCells_Y;i++)
    {
        for(int j=0;j<noOfCells_X;j++)
        {
            g.fillRect(x, y, cellSize,cellSize);
            x = x+CELLSPACING+cellSize;
        }
        x = initCoord.getX();
        y = y+CELLSPACING+cellSize;
    }
}
```

Listing 4. Rendering cells using Java Swing's Graphics methods

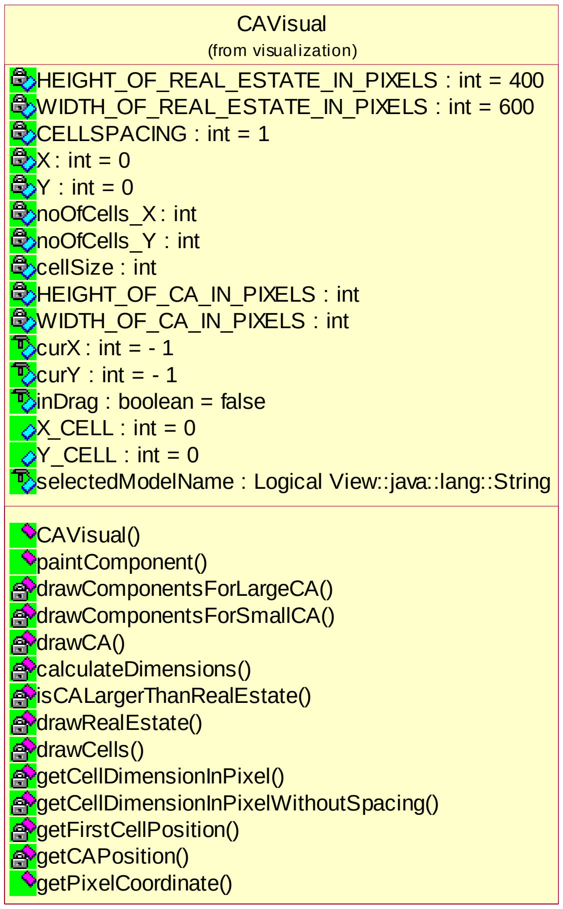
Specializees are rendered the same way using a layered approach in which the base CA is first rendered and then the specialized cells are rendered on top of the base cells using the same methods.

For the Instance Template Models,specializees are not rendered as it is only at instantiation time that specializees have definite locations within an FoR.

This algorithm is computationally a polynomial time function which depends on the size of the CA. In case of very large CAs individual cells are not rendered but the CA is rendered as a solid rectangle thereby reducing it to O(1) time execution.

4.2.3.2 Class Diagram

The CAVisual Class shown in figure 42 represents the visualization of the CA and contains all the methods and algorithms described above. It extends the JPanel class provided by the Java Swing API and so is able to leverage the Graphics methods and PaintComponent methods which allow for platform independent visualizations.



```
CAVisual
(from visualization)

HEIGHT_OF_REAL_ESTATE_IN_PIXELS : int = 400
WIDTH_OF_REAL_ESTATE_IN_PIXELS : int = 600
CELLSPACING : int = 1
X : int = 0
Y : int = 0
noOfCells_X : int
noOfCells_Y : int
cellSize : int
HEIGHT_OF_CA_IN_PIXELS : int
WIDTH_OF_CA_IN_PIXELS : int
curX : int = - 1
curY : int = - 1
inDrag : boolean = false
X_CELL : int = 0
Y_CELL : int = 0
selectedModelName : Logical View::java::lang::String

CAVisual()
paintComponent()
drawComponentsForLargeCA()
drawComponentsForSmallCA()
drawCA()
calculateDimensions()
isCALargerThanRealEstate()
drawRealEstate()
drawCells()
getCellDimensionInPixel()
getCellDimensionInPixelWithoutSpacing()
getFirstCellPosition()
getCAPosition()
getPixelCoordinate()
```

Figure 45. The CA Visual Class representing the viewable CA

Three other classes are used in creating the visualizations and representing the coordinate geometry involved. This class structure is shown in figure 43.

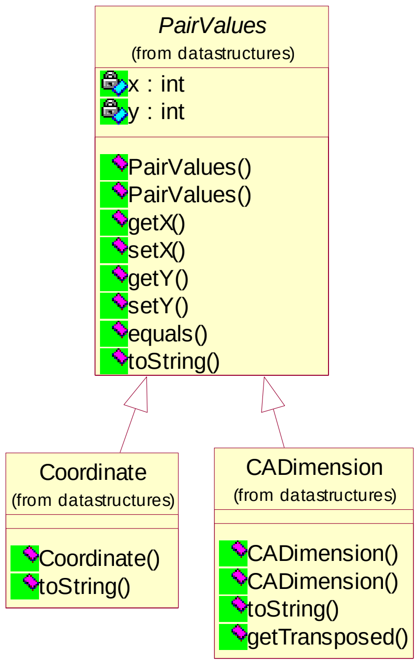


Figure 46. Classes used to represent spatial data

4.2.3.3 Sequence Diagram

The sequence diagram in figure 43 describes the method calls which implement the above discussed algorithms. If the CA is a large one then the drawCA () method is invoked instead of the drawCells() method which in turn would call drawComponentsForLargeCA instead of drawComponentsForSmallCA. The PaintComponent() method call is made by the Java Runtime every time there is an event or when he screen needs to be repainted.

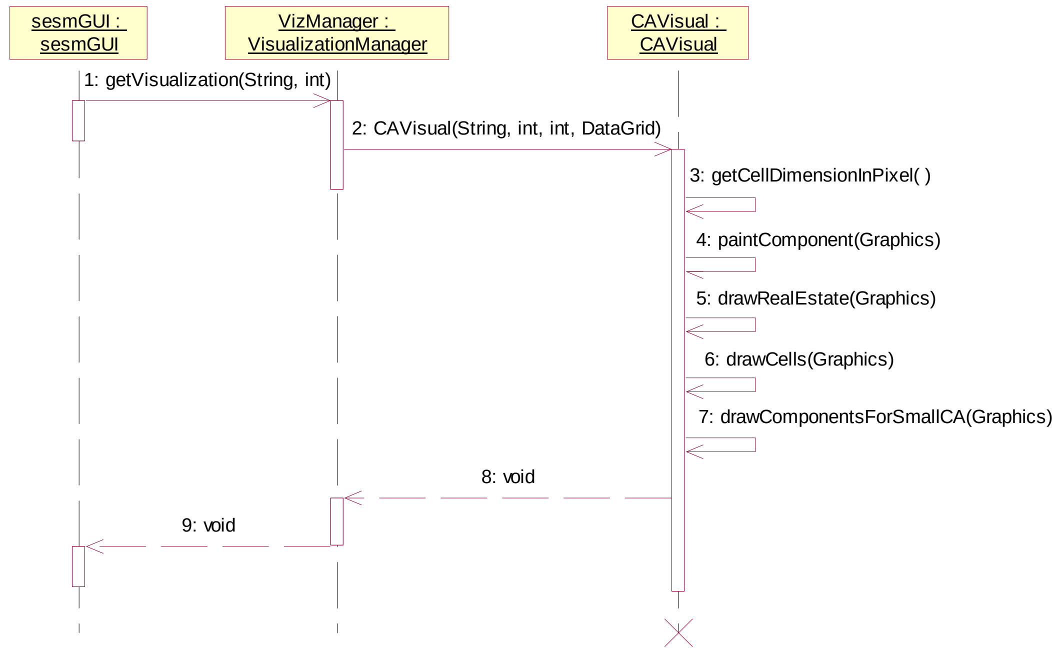

sesmGUI : sesmGUI
VizManager : VisualizationManager
CAVisual : CAVisual
1: getVisualization(String, int)
2: CAVisual(String, int, int, DataGrid)
3: getCellDimensionInPixel( )
4: paintComponent(Graphics)
5: drawRealEstate(Graphics)
6: drawCells(Graphics)
7: drawComponentsForSmallCA(Graphics)
8: void
9: void

Figure 47. Sequence Diagram for CA visualization

4.2.4 Model Export

CoSMoS-CA also support exporting the models created visually into simulatable code. Currently, DEVS is the simulation specification supported and the models are exported in the Java implementation of DEVS called DEVSJAVA. In particular, to run simulations for cellular automata the models are exported to cellular DEVSJAVA code.

4.2.4.1 Algorithm

When the modeler selects a CA model in CoSMoS-CA and exports to cellular DEVS code, 3 models are created:

A model for the cell which has the state variables defined called the Cell model

A model which represents the CA as a collection of cells called the Cellspace model.

A “wrapper” model which extends viewableDigraph into which the Cellspace is added so that the CA may be simulated. This model is called the Container Model. This is a standard DEVS coupled model.

In CoSMoS-CA when a CA is exported to DEVSJAVA code, three .java files are formed:

Model_<Cell Name>.java- corresponds to the container model

Cellspace_<Cell Name>.java- corresponds to the cellspace model

Cell_<Cell Name>.java- corresponds to the cell model

Currently these models are exported to the java export path for CoSMoS.

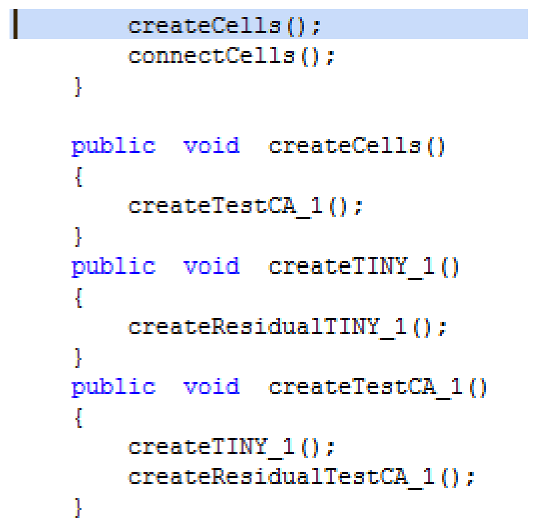
As each cell in the CA,whether it is specialized or not, is homogeneous, it has the same attributes, only the values for them change. Hence while creating the cell model a constructor is created which has all the attributes and in the constructor are assigned to the instance variables with the same name and type. This provides a way to initialize cells differently without changing the structure which remains the same for every cell, thereby ensuring homogeneity. Exporting specializations is done using a complex algorithm which partitions the task of creating cells for specialized regions to nested method calls. The algorithm basically has 2 parts; the first part creates the set of nested method calls and the second creates implementations of those methods.Algorithm1 creates method calls using a Breadth-First traversal of the model tree and for every node n, generates a create_<child_name> method for each of its immediate children and a create_Residual method to create the cells in n which have not been further specialized. This is shown in listing 1.

```
Algorithm1

FUNCTION createMethodRecursively(specializeeName)
```

```
    BEGIN

    children = get immediate children of specializeeName;

    FOR EACH child in children

        GENERATE create_child method call

    END FOR

    GENERATE create_Residual_specializeeName method call

    FOR EACH child in children

        CALL createMethodRecursively(child)

    END FOR

END
```

Listing 5. Algorithm to create the specializee structures during model export

This algorithm creates the series method calls. For example for the model tree below the method call sequence generated is shown below.



```
        createCells();
        connectCells();
    }

    public  void  createCells()
    {
        createTestCA_1();
    }
    public  void  createTINY_1()
    {
        createResidualTINY_1();
    }
    public  void  createTestCA_1()
    {
        createTINY_1();
        createResidualTestCA_1();
    }
```

Listing 6. Code snippet created by Listing1

Algorithm2 creates actual implementations of the methods. This algorithm does a Depth First traversal of the model tree and implements the leaf nodes first and then builds the FoR in such a way that the cells which have been created by the leaf node are not re-constructed. This process continues upwards till the root. Algorithm 2 describes this function.

```
FUNCTION createResidualFunctions(specializeeName)
    GENERATE method signature for createResidual_<specializeeName>
    GENERATE "for i" loop for X-dimension of specializee
    GENERATE "for j" loop for Y-dimension of specializee
    children= GET children of specializeeName;
    FOR EACH child in children
        GET start.X,start.Y,width,height
         GENERATE "if" conditions
            => if(i>=startX AND i<=width AND j>= startY AND j<=height) then continue
    END FOR
    GENERATE addCell call with state variable values for specializeeName;
    FOR EACH child in children
            CALL createResidualFunctions (child)
    END FOR
END
```

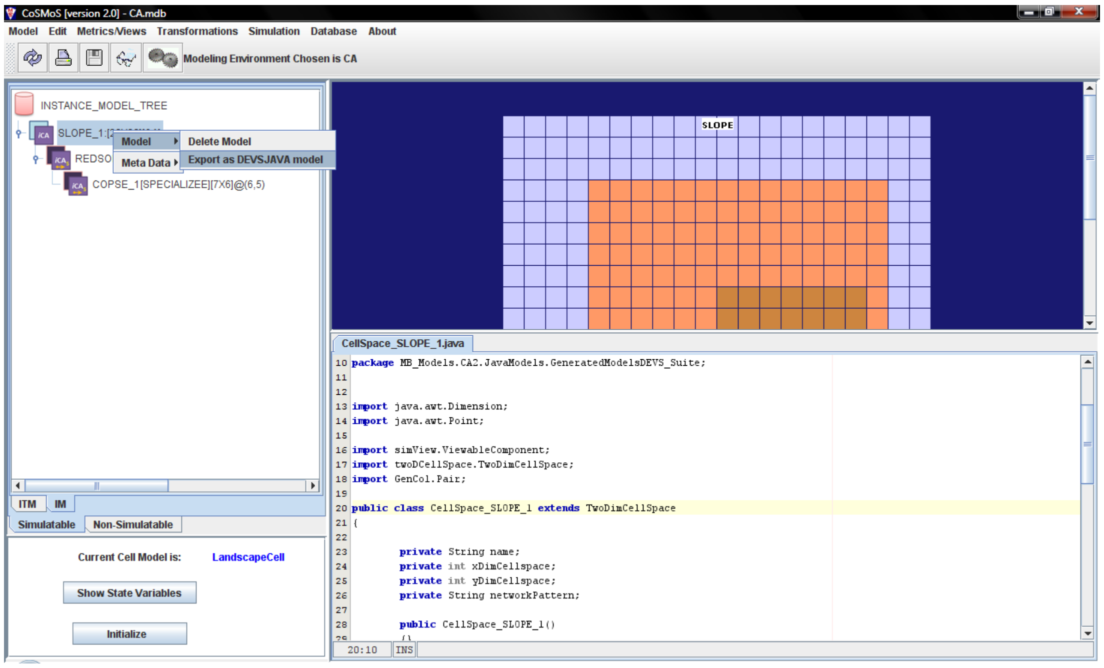
Listing 7. Algorithm to create code for instantiating specialized cells

This algorithm creates the implementations of the residual methods. For example for the model tree shown in listing 4 the method call sequence generated is shown

```
public  void  createResidualTINY_1()
{
    for (int i = 1; i < 2; i++)
        for (int j = 1; j < 2; j++)
        {
            addCell(new Cell_TestCA_1(new Pair(i,j),1));
        }
}
public  void  createResidualTestCA_1()
{
    for (int i = 0; i < 3; i++)
        for (int j = 0; j < 3; j++)
        {
            if(i>=1 && i<2 && j>=1 && j<2)
                continue;
            addCell(new Cell_TestCA_1(new Pair(i,j),1));
        }
}
```

Listing 8. Code generated using Listing3

Once exported these,behavior may be added using the [re-built NetBeans Editor present in CoSMoS.



```
package MB_Models.CA2.JavaModels.GeneratedModelsDEVS_Suite;


import java.awt.Dimension;
import java.awt.Point;

import simView.ViewableComponent;
import twoDCellSpace.TwoDimCellSpace;
import GenCol.Pair;

public class CellSpace_SLOPE_1 extends TwoDimCellSpace
{

        private String name;
        private int xDimCellspace;
        private int yDimCellspace;
        private String networkPattern;

        public CellSpace_SLOPE_1()
```

Figure 48. Exported Code in NetBeans Editor

4.3 Software and Tools

As providing CA-modeling capabilities in CoSMoS is the major part of this research the design reuses several of the existing components. New modules and changes to the existing modules would be made to enable this capability. Java is the programming language used as the existing environment has been completely written in the JAVA programming language. For making the changes to the persistent modeling and simulation framework (CoSMoS), the current database which is MS Access would be used. Java Database Connectivity (JDBC) is the API that is used for accessing the MS Access database. For representing the design of the system and the Entity-Relationship (ER) diagram, and other UML diagrams the Rational Rose Enterprise Edition is used. Eclipse 3.3 is the Integrated Development Environment (IDE) that is used throughout the research.

CHAPTER 5

EXAMPLE MODEL

To demonstrate the capabilities of the CoSMoS-CA environment, we consider an example of modeling and simulating soil erosion on a landscape model. For this example we choose a sample landform, in this case a mountain slope shown in figure 45, and create a CA model representing the slope.

For this problem we consider the slope to be represented by a matrix of 20 x 20 homogeneous LandscapeCells each of which has the following attributes:

R	Rainfall intensity factor
K	a unit-less soil erosion resistance factor based on the percent of sand, silt, clay, and organic matter in the soil. K is scaled from 0 (not erodible) to 1 (highly erodible).
C	a unit-less vegetation erosion protection factor based on the overall ability of different vegetation to hinder raindrops and surface flow, and to bind soil in place. C, like K, is also scaled from 0 to 1. Typical values of C range from 0.9 for bare ground to 0.005 for erosion resistant forest cover.
P	a unit-less erosion prevention practices factor that scales from 0 (perfectly protected) to 1 (not protected).

Figure 49. Example landform

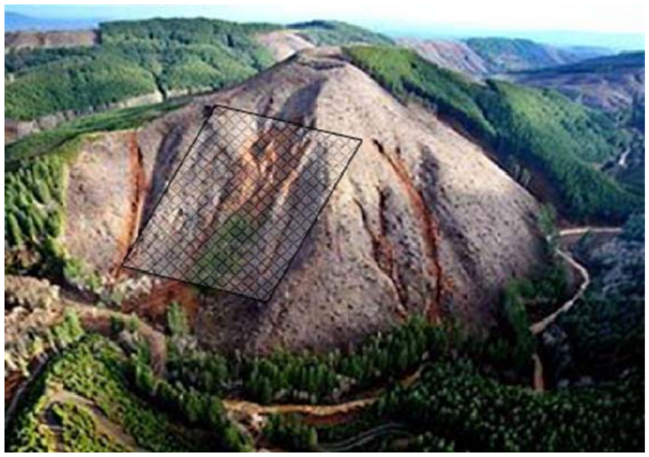


Figure 50. Grid representing the CA for the portion of the landform used in this example

In this example, we assume water is injected on the top of the mountain and simulate the erosion caused by the down flow of the water along the slope.

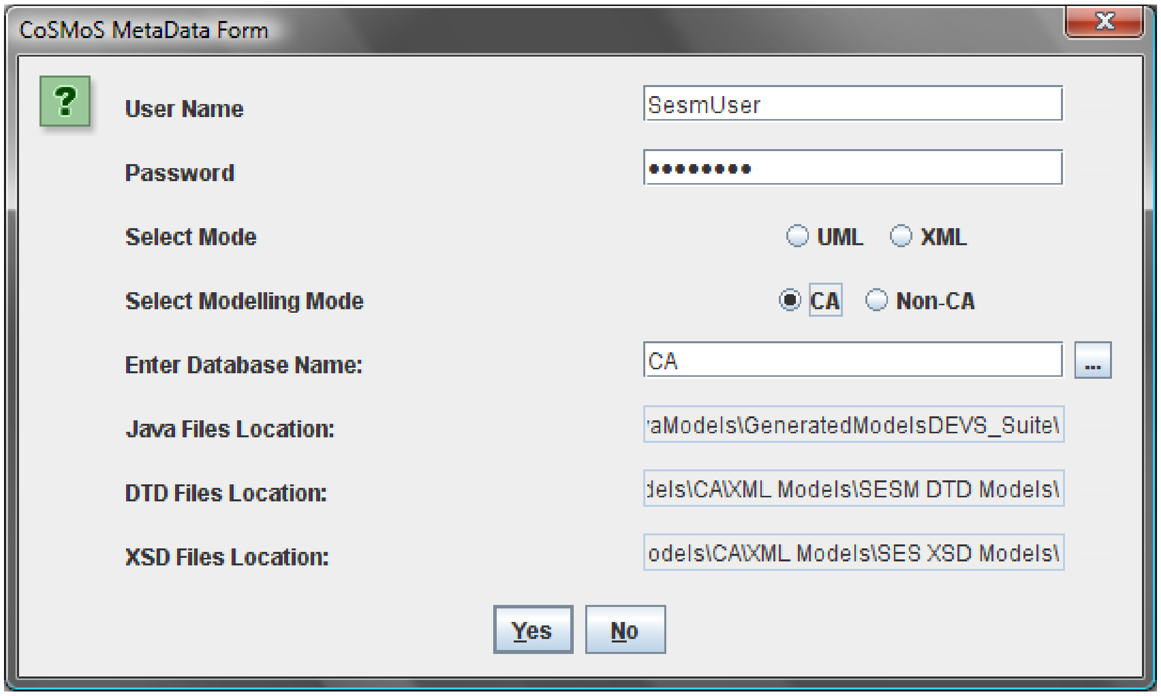
Each cell loses soil which is calculated using the USLE equation:

T =R * K * C * P * LS, where LS is the slope velocity-gradient factor.

Each soil receives the water entity from its neighbors and calculates the soil lost using the above equation and passes it to its neighbors.

5.1 Model Creation

When starting up the environment the user is given the option of selecting an existing database or creating a new database for creating and modifying the models and also which modeling mode to use. The available modeling modes are non-CA and CA. The figure shows the selection of database and establishing the connection for the further processes.


CoSMoS MetaData Form

User Name	SesmUser
Password	●●●●●●●●●
Select Mode	○ UML ○ XML
Select Modelling Mode	[✓ CA] ○ Non-CA
Enter Database Name:	CA
Java Files Location:	'aModels\GeneratedModelsDEVS_Suite\
DTD Files Location:	dels\CA\XML Models\SESM DTD Models\
XSD Files Location:	odels\CA\XML Models\SES XSD Models\

Figure 51. Selecting existing database

As seen, the model can be loaded from an existing database. To see the list of all the existing databases in the working directory, click the button labeled with the "…" next to the "*Enter Database Name:*" the drop down menu lists of all the existing databases.

The environment also allows the user to create new database by simply adding the name of the desired database in the "*Enter Database Name*" text field. A separate folder with the name of the database is created in the file system.

Once the Database is selected the system retrieves all the cell models and prompts the user to select an existing cell model or create a new one. For the current example we select "Create New ". This is shown in the figure 48.

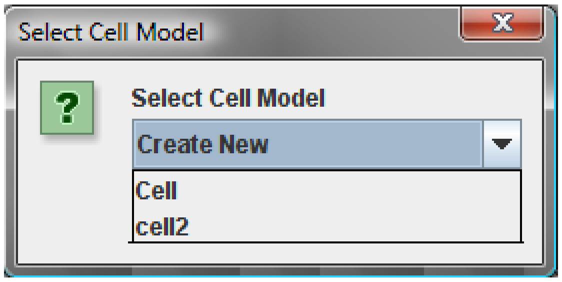
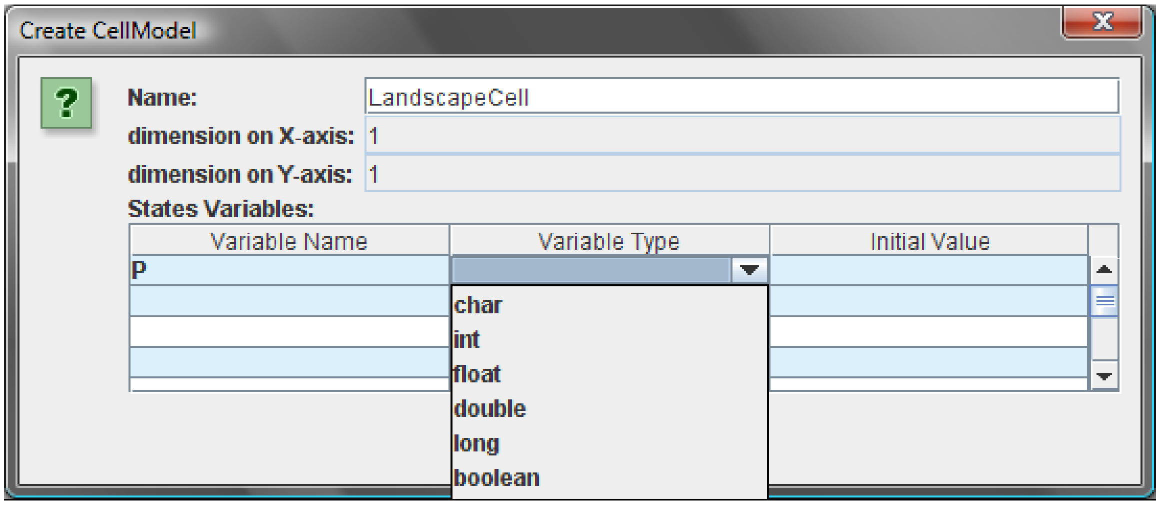


Figure 52. Selecting a Cell model

Next the modeler has to create the cell and specify its attributes. This is done using the dialog shown in figure 49.

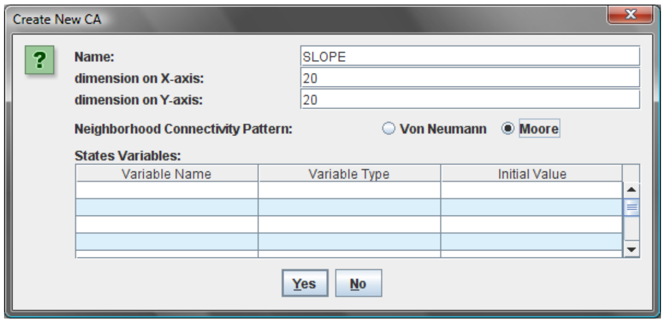


Name: LandscapeCell
dimension on X-axis: 1
dimension on Y-axis: 1

Variable Name	Variable Type	Initial Value
P		

Figure 53. Creating the Cell

This creates the cell and loads the CoSMoS-CA environment for creating families of models using this cell model as the base model. The next step is to create the CA. Click Model→Create New CA and specify the CA structure.

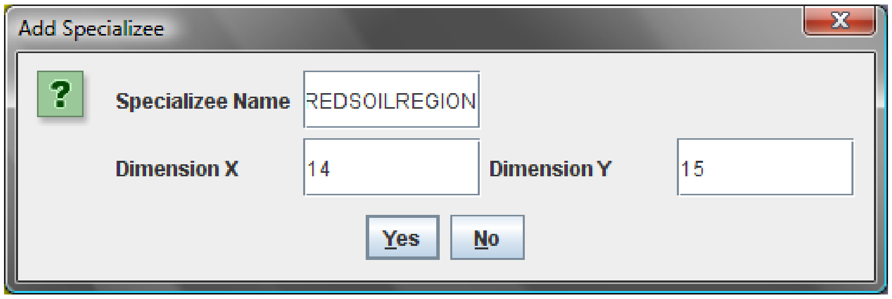


Name: SLOPE
dimension on X-axis: 20
dimension on Y-axis: 20
Neighborhood Connectivity Pattern: Von Neumann [✓ Moore]
States Variables:

Variable Name	Variable Type	Initial Value

Figure 54. Creating the CA in this example called SLOPE

Once the CA is created the user needs to create the specialized regions, in this example create a specializee called REDSOILREGION by right clicking the SLOPE model in the ITM JTree and selecting *Model→Add Specializee* and filling out the dialog shown in figure 51. The dimensions are dependent on domain knowledge and depend on the modeler.



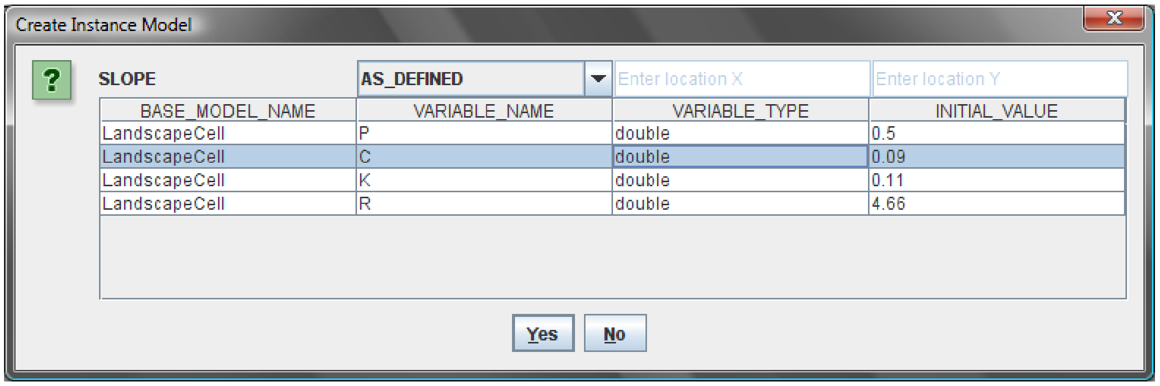
Specializee Name: REDSOILREGION
Dimension X: 14
Dimension Y: 15

Figure 55. Creating the REDSOILREGION specializee

This process is repeated to create another specialized region inside REDSOILREGION called COPSE to represent the region with vegetation.Once done the model is complete.

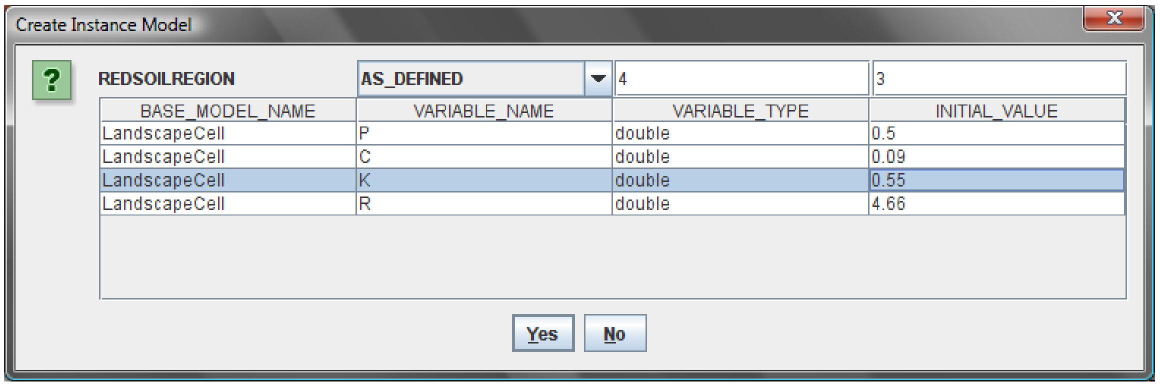
5.2 Creating Instance Model

The model created now is only a template and does not contain any spatial information. This is included by creating an instance model by clicking on the SLOPE model and saying Model→Create Instance. The system shows separate dialogs for SLOPE, REDSOILREGION and COPSE providing the user to specify location and initialization values for the state variables for each specializee apart from the CA itself. Figure 52 shows this.



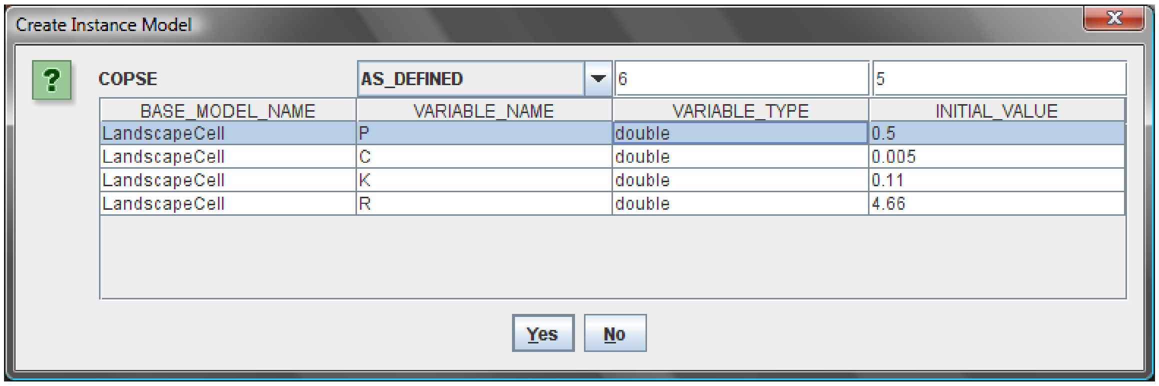
SLOPE AS_DEFINED Enter location X Enter location Y

BASE_MODEL_NAME	VARIABLE_NAME	VARIABLE_TYPE	INITIAL_VALUE
LandscapeCell	P	double	0.5
LandscapeCell	C	double	0.09
LandscapeCell	K	double	0.11
LandscapeCell	R	double	4.66



REDSOILREGION AS_DEFINED 4 3

BASE_MODEL_NAME	VARIABLE_NAME	VARIABLE_TYPE	INITIAL_VALUE
LandscapeCell	P	double	0.5
LandscapeCell	C	double	0.09
LandscapeCell	K	double	0.55
LandscapeCell	R	double	4.66



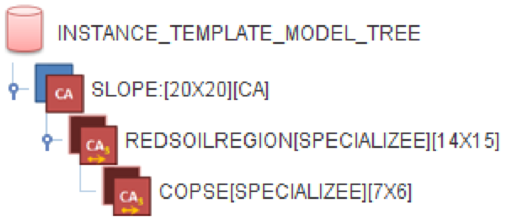
Create Instance Model

COPSE | AS_DEFINED | 6 | 5

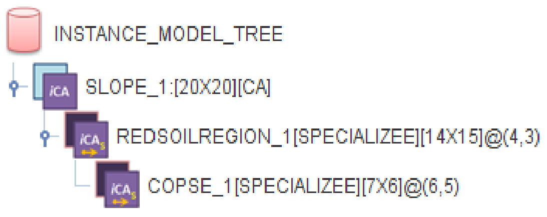
BASE_MODEL_NAME	VARIABLE_NAME	VARIABLE_TYPE	INITIAL_VALUE
LandscapeCell	P	double	0.5
LandscapeCell	C	double	0.005
LandscapeCell	K	double	0.11
LandscapeCell	R	double	4.66

Figure 56. Steps in instantiating the model tree

The values for the state variables for each specializee are domain dependent. Once this is complete the Instance is successfully created and the complete model tree may be viewed in the IM tab and the template structure may be viewed in the ITM tab.



```
INSTANCE_TEMPLATE_MODEL_TREE
  SLOPE:[20X20][CA]
    REDSOILREGION[SPECIALIZEE][14X15]
      COPSE[SPECIALIZEE][7X6]
```



```
INSTANCE_MODEL_TREE
  SLOPE_1:[20X20][CA]
    REDSOILREGION_1[SPECIALIZEE][14X15]@(4,3)
      COPSE_1[SPECIALIZEE][7X6]@(6,5)
```

Figure 57. Instance Template Model Structure

Figure 58. Instance Model Structure

The IM provides a visualization of the created CA when the corresponding node is clicked on the IM JTree. The cells of the FoR are shown in light blue and the specialized cells are shown in red. Currently the visualization only shows one level specializations. These visualizations are shown in figure 55.

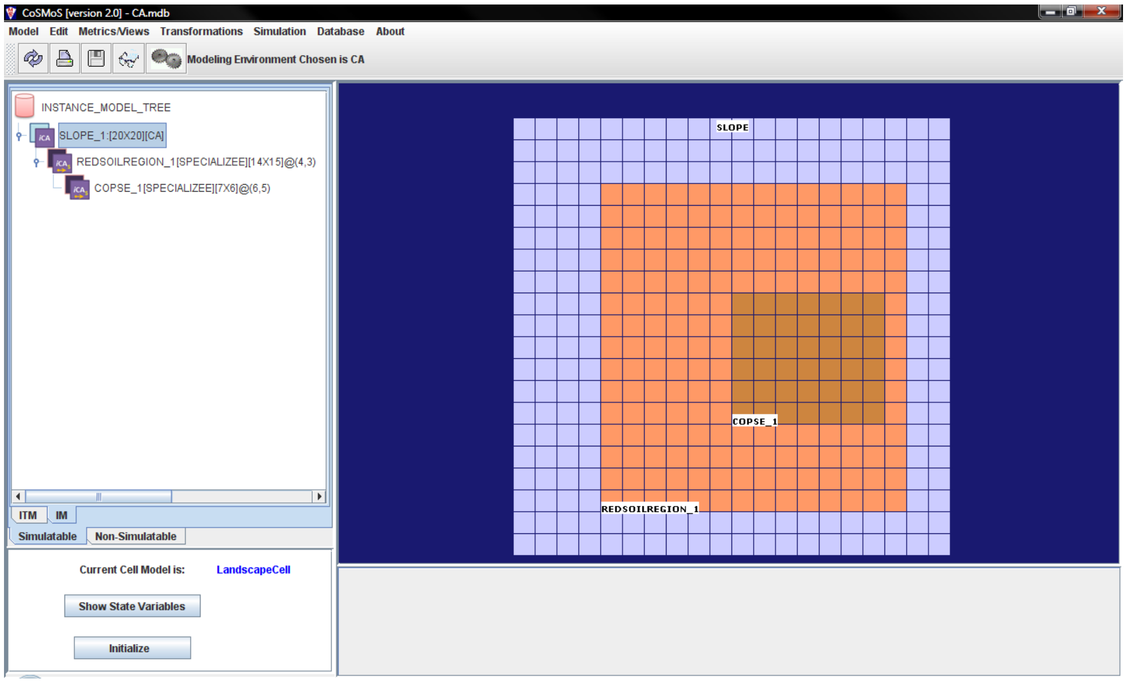
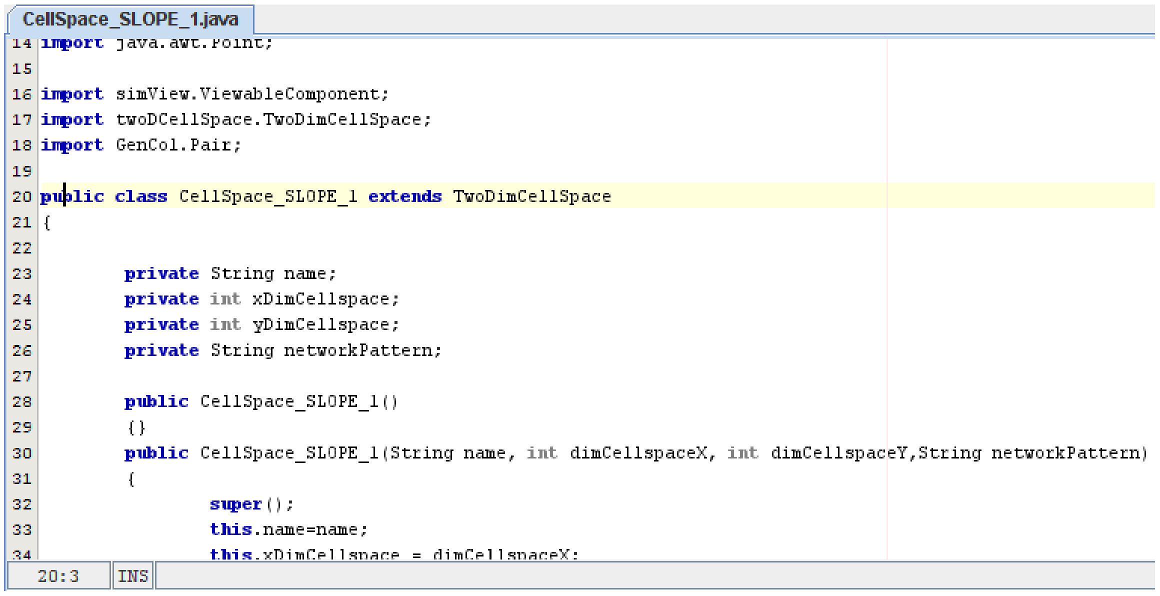

CoSMoS [version 2.0] - CA.mdb
Model Edit Metrics/Views Transformations Simulation Database About
Modeling Environment Chosen is CA
INSTANCE_MODEL_TREE
SLOPE_1:[20X20][CA]
REDSOILREGION_1[SPECIALIZEE][14X15]@(4,3)
COPSE_1[SPECIALIZEE][7X6]@(6,5)
ITM IM
Simulatable Non-Simulatable
Current Cell Model is: LandscapeCell
Show State Variables
Initialize
SLOPE
COPSE_1
REDSOILREGION_1

Figure 59. Visualization of the SLOPE model

5.3 Adding Behavior

Once the instance model is created, the next step is to export the model into simulation code so that it can be simulated. As mentioned earlier CoSMoS-CA currently supports export into cellular DEVS code. The model can be exported clicking on the instance model to be exported and right clicking and selecting Model→Export as DEVSJAVA model. The exported models are placed in the folder for exported Java files which is by default MB_Models\CA\JavaModels\GeneratedModelsDEVS_Suite from the CoSMoS-CA home directory.The exported code may be viewed in the CoSMoS NetBeans Editor by clicking the instance model and clicking the "*View Current Code*" button under the Metrics Menu item.

The code loads in the editor screen as shown in figure 56. Using this, the modeler can add model specific behavior and complete the model's definition.



CellSpace_SLOPE_1.java

```
import java.awt.Point;

import simView.ViewableComponent;
import twoDCellSpace.TwoDimCellSpace;
import GenCol.Pair;

public class CellSpace_SLOPE_1 extends TwoDimCellSpace
{

        private String name;
        private int xDimCellspace;
        private int yDimCellspace;
        private String networkPattern;

        public CellSpace_SLOPE_1()
        {}
        public CellSpace_SLOPE_1(String name, int dimCellspaceX, int dimCellspaceY,String networkPattern)
        {
                super();
                this.name=name;
                this.xDimCellspace = dimCellspaceX;
```

20:3 INS

Figure 60. Exported source code in the NetBeans Editor

5.4 Simulating the Model

The exported models must be simulated on Cellular DEVS separately as currently CoSMoS-CA's inbuilt DEVS-Suite simulator does not include the Cellular DEVS API. Once Cellular DEVS's simulation viewer called SimView is launched the modeler loads the model by clicking the configure button on and including the package with the exported files. The loaded models are displayed on SimView as shown in figure 57.

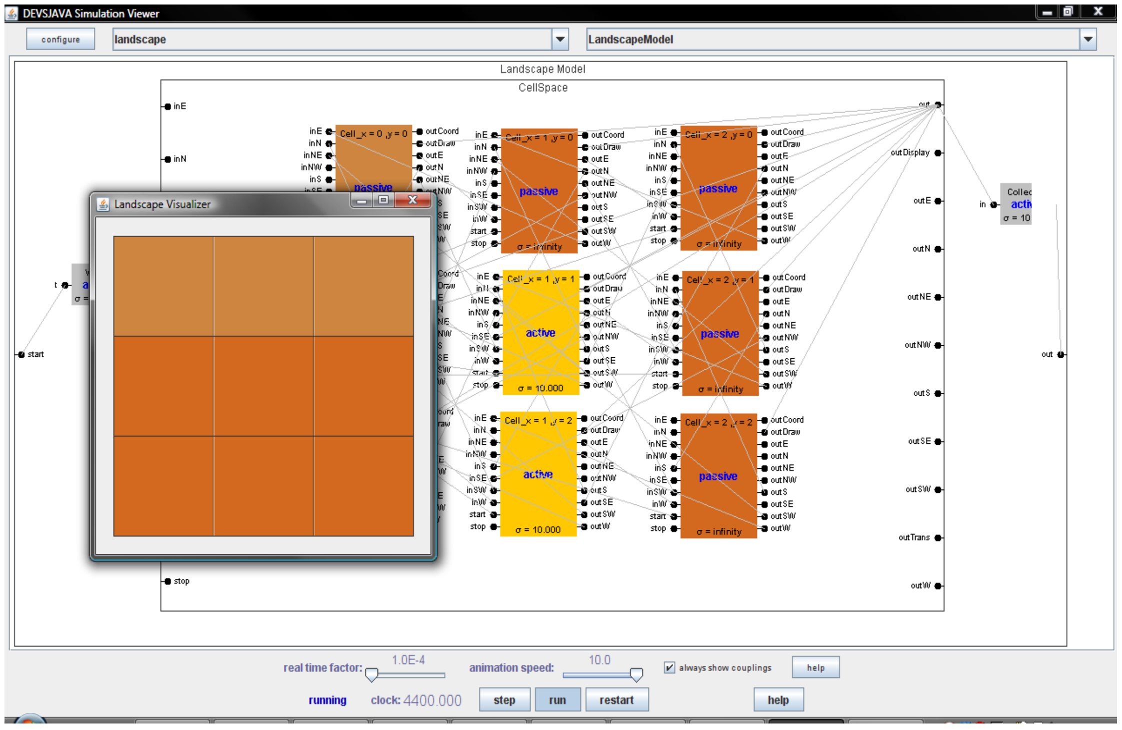


Figure 61. Loaded model in the Cellular DEVS' SimView viewer

CHAPTER 6

CONCLUSION AND FUTURE WORK

In this thesis, a novel capability for modeling structures of Cellular Automata has been introduced and incorporated into CoSMoS, a visual, persistent, and component-based modeling environment. The extended environment supports, homogeneous geo-spatial specifications of CAs by introducing a *spatial specialization* relationship into the CoSMoS's Instance Template Model and Instance Model. Modelers can define any number of cells to be instantiated in terms of their desired spatial arrangements. Furthermore, although the current environment does not support modeling behavior of cells, it supports specifying states for the cells and independently initializing one or more cells for each CA. Several tools exist to create CAs, but none of them can support *visual* development of *families* of model structures. Using this flexible, domain-neutral modeling environment, modelers can develop partial CA models which are specific to the modeler's domain of interest. Thus, multiple CA representations can be visually created and manipulated. The CA models are stored in relational databases and thereafter can be exported into target simulation code. The forward engineering process starting from model specification and ending with code generation is achieved by extending the CoSMoS model process development. The key features of the CoSMoS are demonstrated through modeling of a typical soil erosion process. The environment can be used to develop CA models for a variety of application domains.

In future work, the CoSMoS environment may be extended to support multi-dimensional CAs (greater than 2 spatial dimensions) and composable CAs. Future work also includes integration of one or more simulators in CoSMoS for executing Cellular Automata models. A simulator is needed to allow animation of the cells which vary their colors during simulation execution. It is also useful to support other forms of data visualizations such as time-based trajectories. Currently CoSMoS supports exporting source code for Cellular DEVSJAVA. Future work could support code generation for other kinds of component-based cellular automata simulators. Another desirable feature is to support developing domain-specific models that have pre-built structures and behaviors. Future work could be undertaken in making the UI more interactive by supporting visual manipulation of the individual cells. Furthermore, having a GUI-based capability to add behavior could make the environment more error-free and easier to use. Drag and drop may also be implemented so that modelers, for example, can move specialized regions using the mouse and click instead of entering spatial coordinate information.

REFERENCES

[1] Arizona Center for Integrative Modeling and Simulation (2007). DEVSJAVA. http://www.acims.arizona.edu/software.shtml.

[2] Bendre, S., Sarjoughian, H.S. "Discrete-Event Behavioral Modeling in SESM: Software Design and Implementation". *Advanced Simulation Technology Conference* (2005): pp. 23-28.

[3] Conway, John Horton. "Mathematical Games". *Scientific American* (1970): pp.120-123.

[4] CoSMoS, http://sourceforge.net/projects/cosmosim/. visited April 2009.

[5] DEVS-Suite, http://sourceforge.net/projects/devs-suitesim/. visited January 2009.

[6] Fu, T.S. Hierarchical Modeling of Large-scale systems using Relational Databases. Master's thesis, Computer and Electrical Engineering Dept., University of Arizona, Tucson, AZ. (2002).

[7] Kim, S., Sarjoughian, H. S., and Elamvazhuthi, V. "A Simulator for Visual Experimentation and Behavior Monitoring". *Spring Simulation Multi-conference*, San Diego, CA. (March 2009).

[8] Mayer, G., Sarjoughian, H. S. "A composable discrete-time cellular automaton formalism". In H. Liu, J. Salerno, and M. Young, editors, *Proceedings of the First International Workshop on Social Computing, Behavioral Modeling, and Prediction* (April 2008), pp. 187-196.

[9] Mohan, S. Measuring Structural Complexities of Modular, Hierarchical Large-scale Models. Master's thesis, Computer Science and Engineering Dept., Arizona State University (2003).

[10] Sarjoughian, H. S., Flasher, R. "System Modeling with Mixed Object and Data Models". *DEVS Symposium, Spring Simulation Multi-conference*, Norfolk, VA, USA. (2007), pp. 199-206.

[11] Sarjoughian, H. S., Elamvazhuthi, V. "CoSMoS: A Visual Environment for Component-Based Modeling, Experimental Design, and Simulation". *2nd International Conference on Simulation Tools and Techniques,* invited paper, Rome, Italy. (March 2009), pp. 1-9.

[12] Sarjoughian, H. S. "A scalable component-based modeling Environment Supporting Model Validation" *39th Interservice/Industry Training, Simulation, and Education Conference*, Orlando, FL (November/December 2005), pp. 1-11.

[13] Wilensky, U "NetLogo HomePage". NetLogo. http://ccl.northwestern.edu/netlogo/, Center for Connected Learning and Computer-Based Modeling, Northwestern University. Evanston, IL. (1999).

[14] Wainer.G, Liu Q. "Tools for Graphical Specification and Visualization of DEVS Models". *Simulation Transactions,* Volume 85, No. 3, pp. 131-158, (March 2009).

[15] Wolfram, Stephen. *The Mathematica Book*. s.l. : Wolfram Media/Cambridge University Press 2000.

[16] Zeigler, B., H. Praehofer, T. Kim. *Theory of Modeling and Simulation: Integrated Discrete Event and Continuous Complex Dynamic Systems* (2 ed.). San Diego,California: Academic Press 2000.