HTW des Saarlandes
Wintersemester 2012-2013
Projektarbeit

Domine Specific Language

Dozent: Prof. Dr. Ralf Denzer

Teilnehmer:

Manhal Anwar 3479129

Gilles Baatz 3536491

Sergej Herzog 3502031

Bo Lin 3565556

Daniel Meiers 3538990

Jochen Palz 3479374

David Rupp 3502236

Oliver Wolf 3502007

31. March 2013

Inhaltsverzeichnis

Abbildungsverzeichnis

1 abstract

2 introduction

2.1 Background

Modern information and communication technologies are widely used in almost every field. It has also influenced the environmental science.

2.2 Project Goals

A more universal approach to describe environmental models is needed. A domain specific language was proposed as a solution to this need. A plain DSL will not be enough to solve all problems involved in describing an environmental model in such ways that it can be fully described and exchanged without additional information. The following features were defined as necessary for a language/system that allows to fully describe and exchange models:

- Domain-specific data structures
- Rich model interfaces
- DSL handling typical operations
- Support for different modeling paradigms and frameworks
- Account for modeling uncertainty
- Model transparency and defensibility of results

After the first step of information gathering the group decided to organize the features by priorisation and practicability, to decided which goals are achievable in the given time.
The following order was defined:
priority 1

- Domain-specific data structures
- DSL definition with typical operations
- Account for modeling uncertainty

priority 2

- Rich model interfaces
- Model transparency and defensibility of results

priority 3

- Support for different modeling paradigms and frameworks

As a result of the lack of practical modelling experience the group decided that an universal approach of these feature is not practical. A more practical approach was decided. To gather experience a concrete model will be implemented with different frameworks and then a DSL will be defined with the necessary expressions to implement this model including the ontologies for the data used in the model. With the time given the other features mentioned in the priority list above will only be worked on in a conceptual way. A detailed description of the project proceeding is shown in the paragraph.

2.3 Project Proceeding

2.3.1 Step 1 - Information Phase

In this phase the project participants have to do literature research on environmental modelling in general, and the state of the art of modeling environments

in special. Several modeling environments are under closer examination to get an idea of how the modeling process looks like and what features will be needed in a DSL.

2.3.2 Step 2 - Designing an DSL

With the gathered information from step 1, a Domain specific language (or something alternative) shall be designed that enables modelers to define their models in an intuitive way. This step shall be based on sample model (not to complex, not to simple). And just for one modeling paradigm. The decision for a concrete example must be chosen from a wide used modeling paradigm.

- *Step 2.1 - Ontology based data types*
  The first step is to find out if there are already ontologies witch define datatypes that can be used in models. If there are, they must be analysed for their value for this project. Next a mechanism is needed to dynamically import data types, characterized by ontologies, to parametrize the dsl.
- *Step 2.2 - DSL development*
  With the experience gathered in the former steps a DSL must be defined with the necessary capability to implement the example model.The modeling uncertainty problem must be considered when implementing the DSL and the data types.
- *Step 2.3 Concrete Implementation*
  Some kind of toolchain will be necessary to implement the concrete model with the defined DSL and data types definitions.

2.3.3 Step 3 - Conceptual work

After the implementation conceptual work will be done on the features with priority 2 and 3.

3 Research Phase

3.1 Current Modeling Environments

3.1.1 Simile (Simile Visual Modelling Environment)

Simile is a visual modelling environment, developed originally for the dynamic modelling of ecological and environmental systems, which supports a wide range of ways of representing space. Therefore, it addresses both of the above issues in a single environment.

Visual modeling environment

- Build model without learning programming
- Display concept & relationship using diagram, for example (stocks, flows and influences)
- GUI for simulation control & specifying inputs
- Visualize & compare the model behaviors in graphs & tables

What can we use it for?

- Rate expressions for biological, chemical, or physical processes (differential equations), for example: Mass balance or accounting, Growth dynamics, Respiration
- Scenario analysis, for example: Effect of flow velocity on gas exchange between river-atmosphere

Technical implementation:

Architecture

The following diagram shows the main components of Simile, and how they relate to each other.

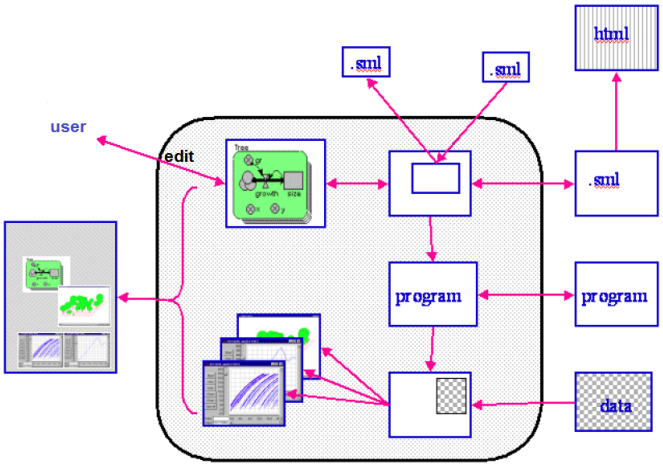

Abbildung 3.1: Simile Architecture

In the middle-left, the user interacts with Simile through a graphical user interface (GUI) to edit the model diagram and equations. In the process, Simile builds up an internal representation of the model, which may contain submodels. The model or a submodel (submodel will be explained later) can be saved to file (.sml extension) (arrows going to the top and to the right). Also, a model or a submodel can be loaded from file (arrows coming in from top and right).

To run a model, the internal representation is used to generate a program in tcl (Tool command language) or C. (In fact, not shown here is that in C Simile first generates the source code, then compiles it as a DLL (Data Definition Language)). This program can also be saved to file, it is possible to run a previously-built model as a stand-alone executable without having access to the Simile environment. The program may then need to be combined with external data sets, containing parameter and tabulated values. After run, Simile calls on various display tools to show the results of the simulation. The displayed results of the simulation may then be exported, along with the model diagram, to produce a postscript file that can be used to produce a handout or poster about the modelling exercise.

At the top-right of the diagram, you will notice that the saved model can be used to generate an html description of the model. This is handled by a program

(written in Prolog) that is totally independent of Simile itself. The html generator is the first of many such tools, and ultimately we envisage that many groups around the world will be writing their own for processing Simile models in new and useful ways.

Properties of the modeling:
Simile has a number of features

Visual modelling:

Simile supports a two-phase approach to model construction. The first involves the drawing of diagrams that show the main features of the model and the second involves fleshing-out the model-diagram elements with quantitative information on the relevant values and equations.

System Dynamics:

Simile allows models to be formulated in System Dynamics terms, as compartments (stocks, levels) whose values are governed by flows in and out. This can be considered as a visual language for representing differential-equation models, with a compartment representing a state variable, and the rate-of-change being the net sum of inflows minus outflows.

Disaggregation:

Simile allows the modeller to express many forms of disaggregation: e.g. age/size/sex/species classes. This is achieved by defining how one class behaves, then specifying that there are many such classes.

Object-based modelling:

Simile allows a population of objects to be modelled. As with disaggregation, model designers state how one member behaves, then specify that there are many such members. In this case, the designer can add in symbols denoting the rules for creating new members of the population, and for killing off existing members. Individual members of the population can interact with others.

Spatial modelling:

It follows that spatial modelling in the system is simply a special form of disaggregation. One spatial unit (grid square, hexagon, polygon, etc.) is modelled, then many such units are specified. Each can be given spatial attributes, such as area or location, and the proximity of one unit to another can be represented.

Modular modelling:

Simile allows any model to be inserted as a submodel into another. Having done this, the modeller can then manually make the links between variables in the two components (in the case where the submodel was not designed to plug into the main model); or links can be made automatically, giving a (plug-and-play) capability. Conversely, any submodel can be extracted and run as a stand-alone model, greatly facilitating testing of the submodels of a complex model.

Efficient computation:

Models can be run as compiled programs. In many cases, these will run as fast as a hand-coded program, enabling Simile to cope with complex models (100s equations; 1000s object instances). While larger or institutional spatial databases are likely to contain millions of object instances, the complexity of modelling, rather than the efficiency of computation, means that dynamic spatial modelling tasks are often more modest in size.

Customisable output displays and input tools:

Simile users can design and implement their own input/output procedures independently. In particular, they can develop displays for model output that are specific to disciplinary norms or other requirements. Once developed, these can be shared with others in the relevant research community.

Declarative representation of model structure:

A Simile model is saved in an open format as a text file (in Prolog syntax). This means that others can develop tools for processing Simile models in novel ways. For example, one group may develop a new way of reporting on model structure, while another may wish to undertake automatic comparison of the structure of two similar models. It also opens the way for the efficient transmission of models across the Internet (as XML files), and for the sharing of models between different modelling environments.

In fact, Simile has no particular spatial representations built into it, rather these are specified by the user in Simile's modelling language and this means that modellers have considerable flexibility in just how space is represented. They are not restricted to some pre-defined spatial framework. One model can include both field and object views, polygonal, rectangular and hexagonal areal units, 3D units (e.g. cubes), and point and linear features, all referenced to a common co-ordinate system. Together with appropriate visualisation tools, this flexibility enables a very wide range of dynamic spatial models to be developed.

The Simile visual modelling environment allow to:

- draw the elements of model, and the relationships between them
- add influences between related variables
- using simple mouse actions, to re-arrange the elements, annotate the diagram or add graphics

System Dynamics symbols

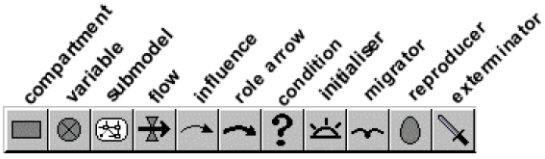

Abbildung 3.2: Simile Symbols

- *compartment :*
  The compartment symbol is used to represent a quantitative state variable
- *Variable :*
  A variable is used to hold one or more values. The value or values come from a mathematical expression
- *Submodel :*
  In Simile, a submodel is a round-cornered box that encloses a number of model-design symbols, including possibly other submodels
- *Flow arrow:*
  The flow arrow is used to specify a term contributing to the rate of change of a compartment
- *Influence arrow :*
  To say that (A influences B) means that A is used to calculate a value for B
- *Role arrow :*
  Role arrows are usually used in pairs, with each arrow going from a submodel to a submodel. The combination of submodels and role arrows is used to denote an association between the objects represented by the submodels at the start of each role arrow
- *Condition :*
  A condition element is placed inside a submodel to indicate that the existence of the submodel depends on some condition

- *Initial-number :*
  The initial-number element is used to specify the initial number of members in a population of objects

- *Migrator :*
  The migrator element is used to specify the rate at which new members of a population are created

- *Reproducer :*
  The reproducer element is used to specify the rate at which each member of a population creates new members

- *Exterminator :*
  The mortality element is used to specify the rule for killing off a member of a population of objects

In common with other visual modelling environments, Simile models are built in two phases. First, the modeller produces a diagram. Then, the modeller quantifies the model by entering numeric values and equations for the various components of the model.
The diagramming icons are chosen from the Toolbar of simile, the compartment, flow, variable and influence, are concerned with conventional System Dynamics modelling, see the following diagram

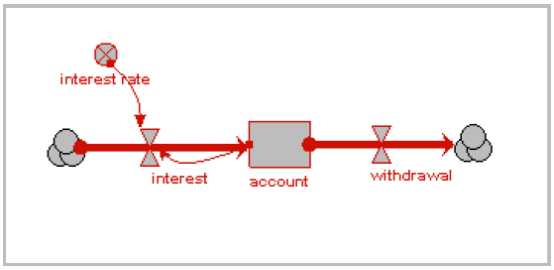

Abbildung 3.3: the model diagram

System Dynamics (SD) is a common dynamic modelling paradigm within the ecological and environmental research communities. A SD model consists of compartments (stocks, levels, storages) connected by flows, with subsidiary variables

for representing parameters and intermediate variables, and influence arrows to show which compartments and variables are used in the calculation of flows and other variables. Essentially, SD is a cosmetic language for defining differential- or difference-equation models: differential equations if the equations are taken to define continuous change; difference equations if the time step is taken to be unity.[1]

Concern following problems of simile

- effort and skill needed to program the models
- the lack of transparency in models implemented as programs
- and the lack of reuseability of models and submodels

3.1.2 esmf (Earth System Modeling Framework)

What ESMF is being developed?
The ESMF is an abbreviation for Earth System Modeling Framework. ESMF is a registered Open Source project for building climate, numerical weather prediction, data assimilation, and other Earth science software applications. These applications are computationally intensive and usually run on supercomputers. The ESMF project is distinguished by community governance and distributed development, and by a diverse customer base that includes modeling groups from universities, major U.S. research centers, the National Weather Service, the Department of Defense, and NASA [01].
ESMF is a component-based software that increases the interoperability of Earth science modeling software developed at different sites and promotes code reuse. The idea of the project is to transform distributed, specialized knowledge and resources into a collaborative, integrated modeling community that operates more efficiently, and is more responsive to societal needs.[2]

[1] http://www.simulistics.com/index.htm and Book: Re-presenting GIS from Peter Fisher
[2] http://en.wikipedia.org/wiki/ESMF and http://www.earthsystemmodeling.org/

Architecture
The components within an ESMF software application usually represent large-scale physical domains such as the atmosphere, ocean, cryosphere, or land surface. The components can create and drive other components.
The software that connects physical domains is called a coupler in the Earth system modeling community. Couplers are used for taking the outputs from one component and transforming them into the inputs that are needed to run another component. In ESMF, couplers are also software components.

Capabilities
In ESMF, user data are represented in the form of data objects such as grids, fields, and arrays. The user data within a component may be copied or referenced into these ESMF objects.
ESMF can associate metadata with data objects. The metadata, in the form of name and value pairs, is grouped into packages. These metadata packages can be written out in XML and other standard formats.

Features
The ESMF differs from other published existing frameworks in the following features:

- Full Fortran 90 interface, partial C/C++ interface
- Fortran 90 Reference Manual and User's Guide
- Runs on most high performance parallel computing platforms, including IBM, many Linux variants, Cray, Compaq, more
- Supports MPI, OpenMP and hybrid user codes
- Free user support
- Active user community[02]

Disadvantages
The ESMF includes user manuals, which should help the beginners to gain entry into the application of the framework. The disadvantage of the manuals is that they are too large and too complex. This complicates the use of the framework.

Although there is FAQ on the website of ESMF, which answers the most frequently asked questions, but there is not a forum where people can share their questions and answers.

3.1.3 oms (Object Modeling System)

What is OMS?
OMS stands for object oriented framework for environmental modeling implemented by java. With OMS, scientists can build scientific model, components efficiently.
It is a collaborative project carried out by US Department of Agriculture and other partner organizations, which are involved in agro-environmental modeling. The goal of OMS is to provide features to the modelers to create inter-operable, scalable models that take full advantage of contemporary computing, management and infrastructure technologies while keeping it simple and intuitive for users.

What is OMS good for?
OMS supports component based software engineering. Models and components are regarded as objects in OMS. The OMS framework introduces a new style of component modelling. No interface should be implemented. OMS uses Meta data with annotations to specify the input and output. This new kind of annotations makes it easier for modelers to identify the interactions of different components. The modelers will have a better overview of each component. Data flow through the components will be clearer for them. The modelers can profit from this feature to create a large component combining different subcomponents to fulfill a complicated task.
The OMS framework provides simulations features for modelers to test the model created by them. Simulations are model applications. It consists of model, input data, output management, execution type, execution environment. OMS provides a DSL to run the simulations. DSL is a mini-language that represents constructs of a given domain. With a DSL, simulations can be created from different tools, for example, IDEs, OMS console. With the simulation feature, the modelers are able to evaluate and improve their models. OMS provides diagram visualization

of the simulation results. Modelers can use this feature to get a deeper insight of the functions of their models.
OMS is designed to support delivery of science relating to agricultural and natural resource management. It is so versatile usable that it can inter-operate with other frameworks of agricultural modeling worldwide.

What is the disadvantages of OMS?

- Not stable. Many exceptions
- Little help documentations
- Costs much time to make OMS work

3.2 problems

3.3 declarative modelling

3.4 semantic approaches

4 toolchain

5 conclusion

6 glossar