COMPONENT-BASED COMPOSITION OF SYSTEM DYNAMICS MODELS

Christian Bauer, Freimut Bodendorf
Department of Information Systems
University of Erlangen-Nuremberg
Lange Gasse 20, 90403 Nuremberg, Germany
{christian.bauer|freimut.bodendorf}@wiso.uni-erlangen.de

KEYWORDS

System Dynamics, Component Technology, Simulation Framework, Model Composition, Software Tool

ABSTRACT

An approach using component technology for the development of System Dynamics models is introduced. At first a brief introduction of System Dynamics and its modeling elements is given. The representation of System Dynamics models in Vensim, an interactive software environment for the handling of such models, is described. On this basis the paper provides a concept for component development and model composition. The concept is basic and independent from any problem or domain specific context. An example depicts the appliance of the concept to a given problem domain explaining the usage of domain specific model design patterns and components. Finally a prototype system to support component-based modeling is described.

PROBLEM

The use of component-based approaches lead to vast improvements in almost every engineering discipline. The list of success stories ranges from classical Industrial production to software development and service manufacturing. The advantages of component technology are well-known. Examples are increased reusability, reduction of complexity, encapsulation of expert knowledge, accelerated production, and the establishment of quality standards.

This paper introduces a concept for the component-based development of System Dynamics models and outlines the potential, that the adoption of component technology can bring to the field of System Dynamics modeling. So far, only few work has been done in this area (Eberlein 1996; Myrtveit 2000; Tignor 2001). Therefore, concepts for modeling components and software environments that support the component-based model composition are rare.

The paper starts with a brief introduction of System Dynamics in general in order to sketch the requirements for the conceptualization of a component-based modeling approach. As the software environment Vensim is used for implementation, its specific representation of System Dynamics is outlined. After that the modeling concept and a prototype system supporting the composition process are described.

SYSTEM DYNAMICS

System Dynamics is an approach that focuses on the analysis of the behavior of complex technical and socio-economic systems. By using simulation models it aims at explaining the system structure that causes the observed behavior. Therefore, the system is decomposed into appropriate elements whereby the causal relationships between the identified elements are revealed.

Modeling Elements

System Dynamics uses mathematical models based on differential equations. Modeling elements are variables and relationships between variables. Relationships are represented by differential equations. Fig. 1 shows the notation that is usually used to visualize the model.

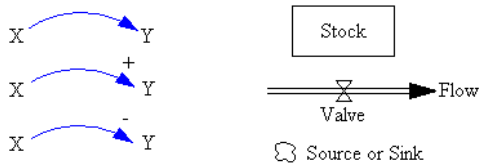

Figure 1: System Dynamics modeling elements

An arrow represents the relationship (causal link) between two variables. The dependent variable is placed at the head of the arrow. Thus, the direction of the relationship is defined. In figure 1 variable Y is influenced by variable X. The formal expression is $Y = f(X)$. Link polarities can be assigned to causal relationships. If $dY / dX > 0$ applies, the link polarity is positive, marked by a plus symbol at the head of the arrow. If the link expresses the relationship $dY / dX < 0$, a minus symbol is used (Sterman 2000).

A variable that accumulates the influences it receives over time is referred to as a stock (see figure 2). A stock gives systems inertia and provides them with memory. The change of state that affects a stock at any point in time is described as a flow. The amount flowing in or out of a stock is controled by a valve. Clouds are used to indicate that the source or the drain of such a flow is outside the model boundaries.

Proceedings 19th European Conference on Modelling and Simulation
Yuri Merkuryev, Richard Zobel, Eugène Kerckhoffs © ECMS, 2005
ISBN 1-84233-112-4 (Set) / ISBN 1-84233-113-2 (CD)

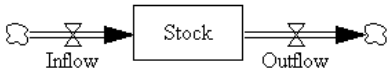

Figure 2: Stock and flow

The mathematical representation is:

$$\text{Stock}(t) = \text{Stock}(t_0) + \int_{t_0}^{t} [\text{Inflow}(s) - \text{Outflow}(s)]ds$$

The value of a stock is computed through integration of the differences between inflow and outflow at any point in time s between a starting point t_0 and the actual point in time t under consideration of the initial state of the stock at t_0 (Sterman 2000).

By combining these modeling elements the structure of the system to be analyzed is given. Feedback loops are in the core of the model development. Causal loops are the most important elements to define the behavior of the system. Decisions, intended to govern the system's behavior, are always part of a causal loop with that system (see figure 3).

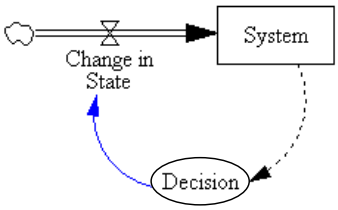

Figure 3: Decision as a feedback process

A decision is made by applying a set of decision rules to the system. Input is information about the state of the system. The decision changes that state of the system. Therefore a decision is part of a causal loop comprising the decision, the initiated change in state and the state of the system. Through simulation it is possible to analyze the behavior of the model over time and thereby approve the appropriateness of the applied decision rule (policy).

Model Representation in Vensim

Vensim is an interactive software environment for the development, simulation, and exploration of System Dynamics models (Ventana 2003). In Vensim models are created either by a text editor or by a sketch editor. The text editor is a general-purpose ASCII-editor that allows the specification of the model's underlying variables and equations. The sketch editor on the other hand provides a graphical user interface to the modeling elements. No matter in which way a model is created, Vensim always stores the model data in a single file. Two basic file formats are available. The format .vmf stores model data as binary code while in .mdl-files the model data is stored as plain text (Ventana 2003). Figure 4 shows the structure of a simple System Dynamics model created with the sketch editor.

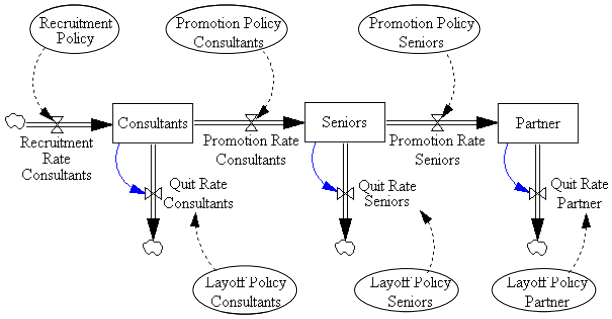

Figure 4: Simplified System Dynamics model capturing the workforce of a consultancy

The model captures the workforce of a consulting company and consists of three stocks, each representing a job level within the organization. The number of staff at each job level at any point in time t is computed through integration of the differences between inflow and outflow during the period t - t_0. Quit rates reduce both, the number of staff at the regarded job level as well as the total number of staff. Promotion rates in contrast shift staff from the preliminary to the next job level. Solely the recruitment of new consultants increases the overall number of staff and thereby offers the possibility to compensate fluctuation. The maximum promotion and quit rates depend on the value of the stock they are related to. Recruitment rate, promotion rate as well as part of the quit rate are subject to management policies that define the target values for these rates. In figure 4 variables that represent such policies are marked by a circle. The values of these variables are either subject to user parameterization or have to be determined through additional models that implement the guiding rules for these policies.

Opening the model in the text editor reveals the structure of the underlying differential equations. Figure 5 shows the model variables, equations, and simulation control parameters in the .mdl-file-format.

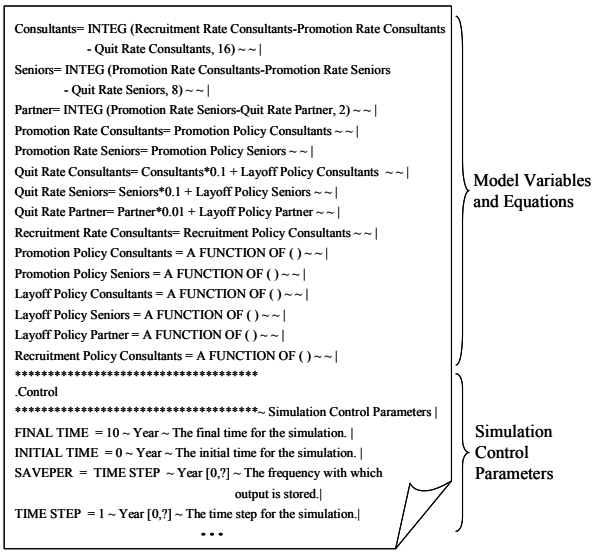

Figure 5: Model variables, equations and simulation control parameters

Each variable is defined by an equation that determines its value. Further it is possible to specify a dimension for the variable and to place some comments. The format has the following structure:

<equation> ~ <dimension> ~ <comment> |

The character "~" separates the elements of the definition, while the character "|" terminates the definition as a whole (Ventana 2003a). E. g., the first line of text shown in figure 5 defines an equation to determine the value of the variable "Consultants". The number of consultants is computed as an integral of the difference between the recruitment rate and the sum of promotion and quit rates at each given point in time. It is defined that the initial value of "Consultants" is 16. The variable has no unit specification and no comments are given. Processing the text line by line is at first straight forward. Several variables are defined that determine the structure of the model. The definitions of variables representing management policies stand out. The right hand sides of these equations all contain the string "A FUNCTION OF ( )". This string, a keyword of the Vensim Modeling Language, indicates that no equation is defined for the given variable (Ventana 2003a, p. 68). Thus the model of figure 5 is incomplete and therefore not ready for simulation. On the other hand this keyword provides an important starting point for the development of model components.

The definition of the model's variables and equations is followed by a section that specifies the simulation control parameters "INITIAL TIME", "FINAL TIME", "SAVEPER" and "TIMESTEP".

The last section of the .mdl-file-format contains so-called sketch information, that is the information needed to compute the graphical representation of the model's structure. Figure 6 shows an excerpt of the sketch information needed to represent the exemplary model.

```
...
\\\---/// Sketch information - do not modify anything except names
V300  Do not put anything below this section - it will be ignored
*View 1
$192-192-192,0,Times New Roman|12||0-0-0|0-0-0|0-0-255|-1--1--1|-1--1--1|96,96,100
10,1,Consultants,302,230,40,20,3,3,0,0,0,0,0,0
10,2,Seniors,492,229,40,20,3,3,0,0,0,0,0,0
10,3,Partner,683,230,40,20,3,3,0,0,0,0,0,0
1,4,6,2,4,0,0,22,0,0,0,-1--1--1,,1|(428,229)|
1,5,6,1,100,0,0,22,0,0,0,-1--1--1,,1|(367,229)|
11,6,268,399,229,6,8,34,3,0,0,1,0,0,0
10,7,Promotion Rate Consultants,399,256,50,19,40,3,0,0,-1,0,0,0
1,8,10,3,4,0,0,22,0,0,0,-1--1--1,,1|(616,229)|
1,9,10,2,100,0,0,22,0,0,0,-1--1--1,,1|(554,229)|
11,10,332,583,229,6,8,34,3,0,0,1,0,0,0
10,11,Promotion Rate Seniors,583,256,50,19,40,3,0,0,-1,0,0,0
12,12,48,166,228,10,8,0,3,0,0,-1,0,0,0
...
```

Figure 6: Sketch data

The beginning of the sketch information is always marked by the string "\\\---///" followed by a comment. The second line starts with a version code that indicates the format of the sketch information and is also followed by a comment. The third line names the view of the sketch and is preceded by the char "*". By the definition of multiple views it is possible to spread the graphical representation of the model over several windows. Thus, exploration and handling of large models is facilitated. The line beginning with a "$" sets the default font and colors of the view. The remaining lines define the objects appearing on the sketch. Each line defines one object following a special format. E. g., the first number sets the object type, the second number is the ID of the object. Objects that represent a variable contain the variable name, whereas an object that represents an arrow contains the ids of the two variables linked together (Ventana 2003a).

MODEL COMPONENTS

A model component is a piece of model that can be used as a building block. Fully-fledged models are developed by linking several model components together. The concept of a model component corresponds in some way to the concept of a class known from object engineering. Thus, a model component is an artifact that encapsulates a well-defined structure. As it uses specified interfaces to interact with other model components, polymorphism is achieved, allowing the substitution of single components. Thereby a component-based model can easily be adjusted to solve a new or modified problem.

Model Component Specification

The model component specification is independent from any problem or domain specific context and defines the general structure that has to be followed by every domain specific model component. Figure 7 depicts the definition of a model component using the notation of an UML class diagram.

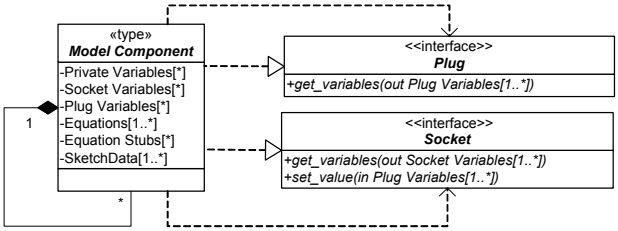

Figure 7: Model component specification

A model component is a piece of model, therefore it contains several model variables, equations and the corresponding sketch data. By aggregation it is possible to built model component hierarchies. The interaction between components is handled by the definition of interfaces ensuring the substitutability of single components. As shown in figure 7 a model component defines two interfaces, a plug interface and a socket interface. At least one of these interfaces has to be implemented, otherwise no interaction is possible.

The plug interface comprises a set of model variables called plug variables. Plug variables are used to connect to the socket variables of another component's socket interface. In contrast to plug variables, whose values are defined through a corresponding equation within the model component, socket variables have only so-called equation stubs, marked by the keyword "A FUNCTION

OF ( )". During assembly the equation stub of the socket variable is replaced by the equation of the corresponding plug variable. User-defined keywords are used to determine whether a model variable is a socket or a plug variable. E. g., for socket variables the string "SO" might be defined, while for plug variables the string "PL" might be suitable. To determine which plug variable implements a given socket variable, the names of corresponding plug and socket variables have to be identical, except their keyword. E. g., a plug variable intended to implement a socket variable named "Recruitment Policy Consultants SO" must be labeled "Recruitment Policy Consultants PL". Figure 8 illustrates the assembly process.

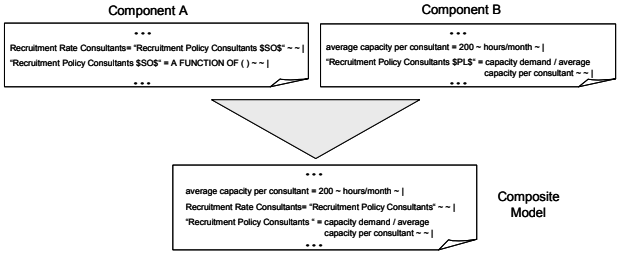

Figure 8: Model composition

Model component A contains the socket variable "Recruitment Policy Consultants SO". This socket variable is used to determine the value of the variable "Recruitment Rate Consultants". As the socket variable only possesses an equation stub, no value can be computed. Component B defines the corresponding plug variable "Recruitment Policy Consultants PL" and provides an equation to determine the variable's value. Once both components are assembled, the equation stub of the socket variable is replaced by the equation of the plug variable and the keywords "SO" and "PL" are deleted to simplify variable names within the composite model.

A similar technique is used to allow multiple instantiation of a component. As variable names have to be unique, it is not possible to use the same variable name in two instances. To solve this problem, the string "#VAR#" is used as keyword in every model variable's name. During component instantiation this keyword is replaced by the ID of the created entity. E. g. the variable "Recruitment Rate Consultants #VAR#" of a given component becomes "Recruitment Rate Consultants 1" for the first instance and "Recruitment Rate Consultants 2" for the second instance. Thereby it is ensured that every variable's name remains unique.

Derivation of Domain-specific Components and Model Design Patterns

Based on the model component specification, domain-specific components are derived. System Dynamics models are usually built to analyze a well-defined problem or phenomenon and provide insight into the inherent structure of the system under investigation. Within each problem domain certain domain-specific concepts exist and domain-specific components are used to capture this knowledge. Further domain-specific model design patterns can be created through the definition of explicit relationships between components. Thereby it is possible to predefine the structure of the prospective composite model, thus additionally guiding and facilitating the model building process. Figure 9 gives an example of a domain-specific design pattern that depicts the structure of a model capturing the collaborative relationships between consultancies.

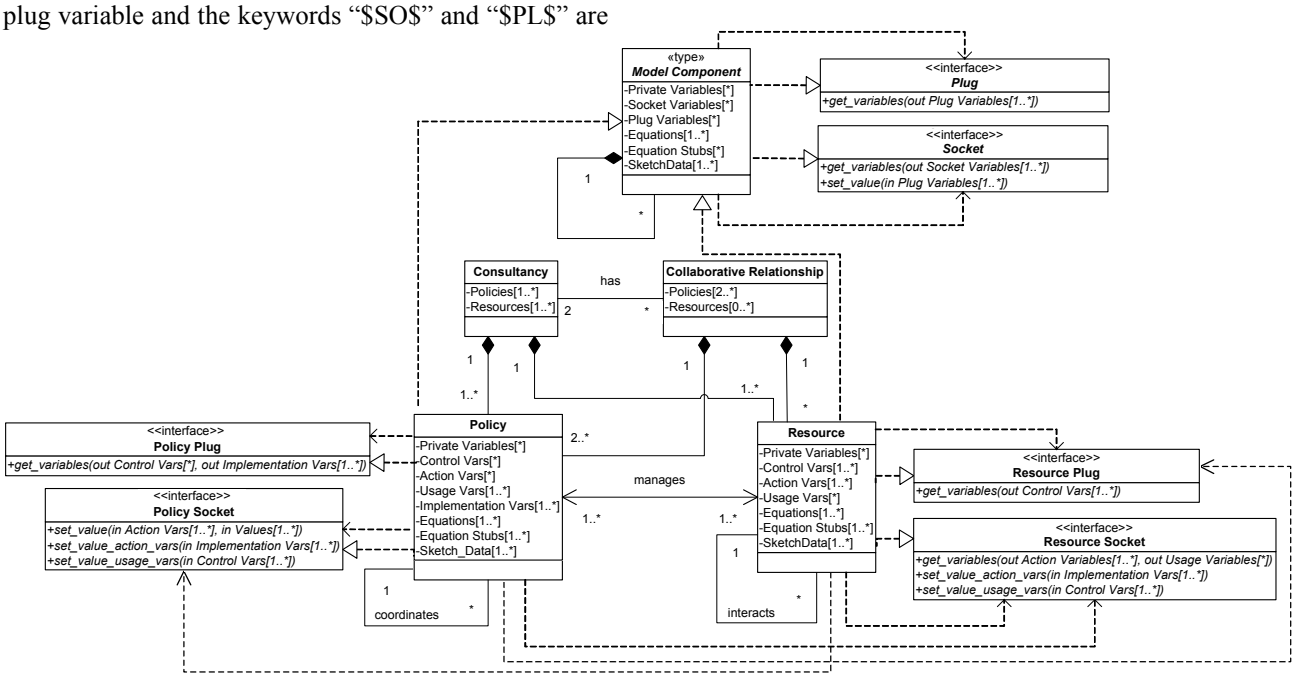

Figure 9: Domain-specific model design pattern including derived components

The design pattern defines two levels of abstraction and four types of components. On the first level the components "Consultancy" and "Collaborative Relationship" are defined. Both are aggregations of the components "Policy" and "Resource" on level two. As they merely act as containers, they don't need to specify separate interfaces, but draw on the interfaces implemented by the components they include. Nevertheless the design pattern defines a relationship between both components, which specifies that two instances of "Consultancy" are needed for a collaborative relationship to exist. Further it is defined that a consultancy at least consists of one policy component and one resource component, while a collaborative relationship aggregates at least two policy components (one of each consultancy involved).

As figure 9 depicts, the components "Policy" and "Resource" implement the model component specification. Besides the necessary plug and socket interfaces they define several new variable types derived from the abstract model component. The declaration of new variable types is done through the specification of corresponding keywords. E. g. the component "Resource" contains private, control, action, and usage variables. Control variables (keyword "CV") are used as plugs and determine which information of the model piece can be used by the sockets of other components. Action variables (keyword "AV") and usage variables (keyword "UV") in turn are socket variables which provide only equation stubs and thereby indicate where the building block depends on input from other model components. The methods of the resource socket interface specify that the value of action variables is set by implementation variables (keyword "IV") while the value of usage variables refers to control variables. The mapping of variable types allows the specification of relationships between components. Thus the relationship "interacts" between resources is implemented by pairs of control and usage variables. The relationship "manages" between a resource and a policy comprises two aspects. First the policy has to define usage variables as part of its socket interface in order to access the resource's control variables. Second it has to define one or more implementation variables which can be used as a plug to implement the resource's action variables.

As a policy component may also define control and action variables itself, it is possible to assign the government of the policy to another, superior policy component. Thereby hierarchies of policies coordinating each other can be created ("coordinates" relationship between policies).

MODEL COMPOSER

While Vensim provides an environment for the development of model components as well as for the simulation and exploration of the composed simulation models, it does not support the process of model composition itself. Therefore a tool has been developed, that on the one hand provides a graphical user interface to facilitate the composition process and on the other hand helps to administrate the used model components, design patterns, and composed simulation models. The tool has been implemented using the Microsoft .NET Framework and the Dynamic Link Library of Vensim.

Architecture

Figure 10 shows the architecture of the prototype.

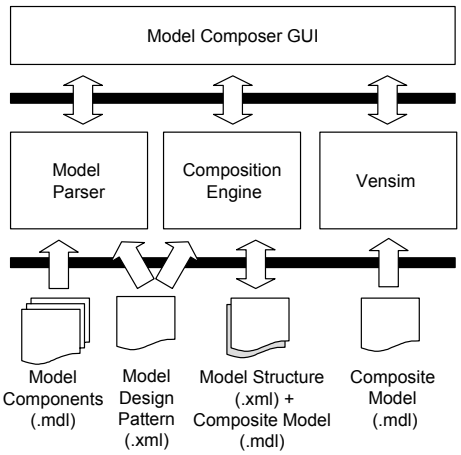

Figure 10: Architecture of the model composer

The application basically consists of three separate modules that are accessible through a common graphical user interface. The model parser is used to analyze the structure of a model component and to create a processable internal representation of it. As a model design pattern specifies the types of valid components and their relationships, it is necessary to load a design pattern into the parser first. After that the .mdl-files of model components are opened and the content is parsed according to the assigned design pattern. The parser uses the specified keywords to access the variable definitions of the pattern and thereby determines the types of the loaded components.

The composition engine implements the routines necessary to automate the assembly of the chosen components. To produce a composite model, the composition engine takes the internal component representations, combines them according to the relationships defined within the design pattern and creates a new .mdl-file for the composite model. Further a .xml-file is created to store the relevant composition information (e. g. the types of components and the number of instantiations). Thereby it is ensured that composite models can be decomposed at any time for further modifications.

In order to facilitate the creation and inspection of model components as well as the exploration and simulation of composed models, Vensim is integrated into the model composer. Thereby frequent manual switch-overs between both applications can be avoided and the usability of the system is enhanced.

Composition Process

Figure 11 shows the graphical user interface of the prototypical model composer.

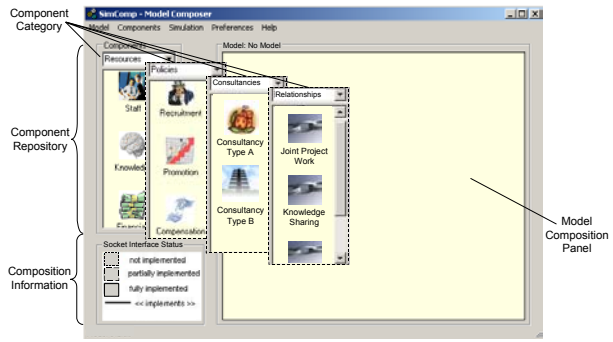

Figure 11: Graphical user interface of the model composer

Available Components are categorized according to their component type. As the exemplary model design pattern shown in figure 9 defines four types of components, the model composer provides four corresponding component categories. Within each category several components exist, that jointly make up the available component repository.

To compose a new model the user simply drags a component from a chosen category and drops it onto the model composition panel. The drag-and-drop-mechanism automatically activates the composition engine, which creates a new instance of the component (replacement of the keyword #VAR#). The composition engine scans the interfaces of all instantiated components on the panel and if a matching pair of socket and plug variables is detected, the involved components are linked together. Figure 12 gives an example of a model being in the composition process.

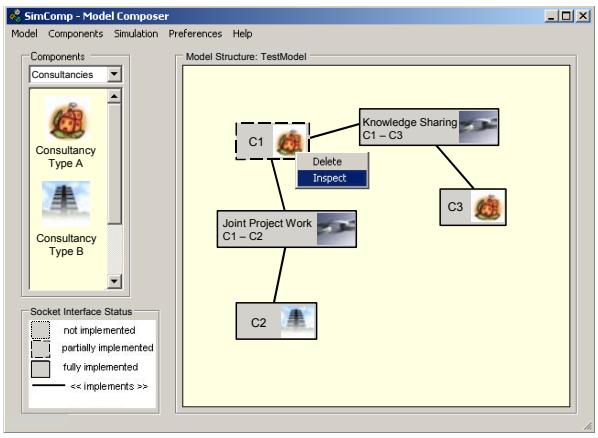

Figure 12: Construction of composite models

The model composition panel comprises two instances of the component “Consultancy Type A”, one instance of “Consultancy Type B” and two instances of relationships. The lines between the instances indicate the links detected by the composition engine. Further the implementation status of each component’s socket interface or, in case of containers, the aggregated status of the socket interfaces of any child instances, is indicated by the type of border line surrounding the instance on the composition panel. A dotted border signals that no variable of the instance’s socket interface or of the socket interfaces of any children is implemented. This is the case when a new component is placed onto the panel, that has no links to already existing instances. A dashed border indicates that some of the socket variables are implemented through corresponding plugs, while a solid border states that the socket interface is fully implemented. If all instances show a solid border, the composition process is finished and the composite model is ready for simulation.

According to the applied model design pattern, the instances shown in figure 12 are all defined as containers. Thus they don’t define their own interfaces but aggregate the interfaces of the components they include. This means the detected links have their root at a lower level of aggregation. The dashed border of instance “C1” indicates that the container includes at least one component whose socket interface is not fully implemented. By selecting the menu item “Inspect” from the context menu of any instantiated component it is possible to explore its content. Figure 13 shows the content of the instance “C1”.

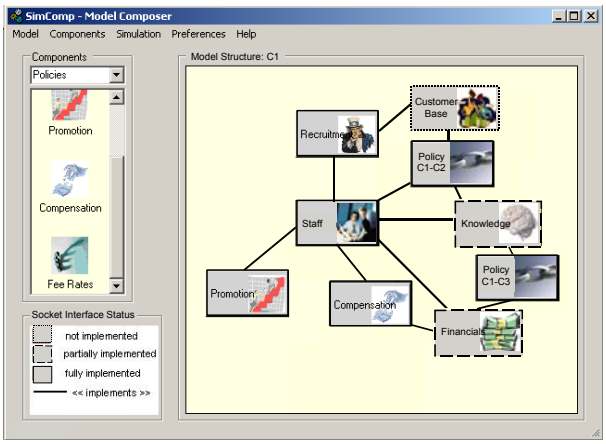

Figure 13: Content of instance “C1”

As defined in the model design pattern, components of the type “Consultancy” are aggregations of policies and resources. Therefore “C1” includes several instances of these two categories. As figure 13 shows, these instances are interrelated. E. g., the instance “Staff”, a component of the type “Resource”, is linked to the instances “Promotion”, “Recruitment” and “Compensation”, which are identified as “Policies” managing this resource. Apparently these three policies fully implement the socket interface of “Staff” (indicated by the solid border of the instance). In contrast to “Staff”, none of the instances on the panel implements the socket variables of “Customer Base” (dotted border). Further the socket variables of the resources “Knowledge” and “Financials” are only partly implemented (dashed border).

The instances “Policy C1-C2” and “Policy C1-C3” establish the links between “C1” and “Joint Project Work C1-C2”, respectively “Knowledge Sharing C1-C3”, which are both instances of the component category “Relationships” and therefore map the component type “Collaborative Relationship”. The applied model design pattern defines this component type as an aggregation of at least two policies, one of each consultancy involved (see section 3.2). This implies that “Policy C1-C2” and “Policy C1-C3” are simultaneously part of two aggregations, “Consultancy” and “Collaborative Relationship”. Nevertheless the inspection of “C1” only reveals the links to components that are embraced by “C1”. To inspect the links of “Policy C1-C2” and “Policy C1-C3” relating to components embraced by “Joint Project Work C1-C2” respectively “Knowledge Sharing C1-C3”, it is necessary to explore the structure of these aggregations.

Although the type of border line of an instance indicates the status of it’s socket interface and the lines between instances depict their interrelations, it is useful to provide more detailed information about the implementation and usage of the interface variables involved. Selecting “Inspect” from the context menu of a component that is not a container displays its interface variables categorized by type. Further the relationships of these variables to corresponding plug, respectively socket variables of other components are shown. Figure 14 gives an example.

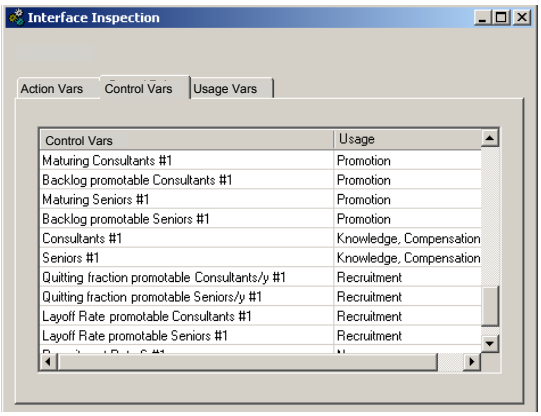

Figure 14: Interface inspection of instance “Staff”

The figure shows the inspection of the instance “Staff”. According to the specification of the component type “Resource”, the interface variables are defined as “Action Vars”, “Control Vars” and “Usage Vars”. Selecting the tab “Control Vars” returns a list of all model variables marked as “Control Vars”. Further the column “Usage” shows, which instances define socket variables that use a certain control variable as plug. E. g., the variable “Maturing Consultants #1” is used by the instance “Promotion”, whereas the variable “Consultants #1” is used by “Knowledge” and “Compensation”.

Figure 15 shows the composed model. The solid borders of the component instances indicates that all interfaces are fully implemented and the model is ready for simulation.

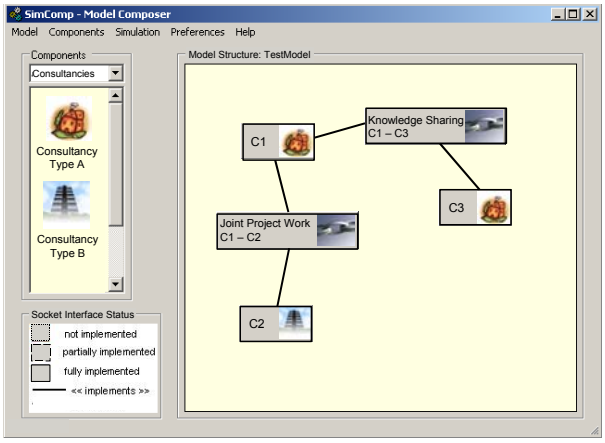

Figure 15: Composed model

Selecting “View Model Structure” from the menu item “Model” calls the Vensim environment to inspect the structure of the composed system dynamics model. The ability of Vensim to handle multiple views within a model is used to keep the graphical representations of the component instances separated. The usage of multiple views reduces complexity and facilitates model navigation. Figure 16 shows the component instance “Joint Project Work C1 - C2” in Vensim.

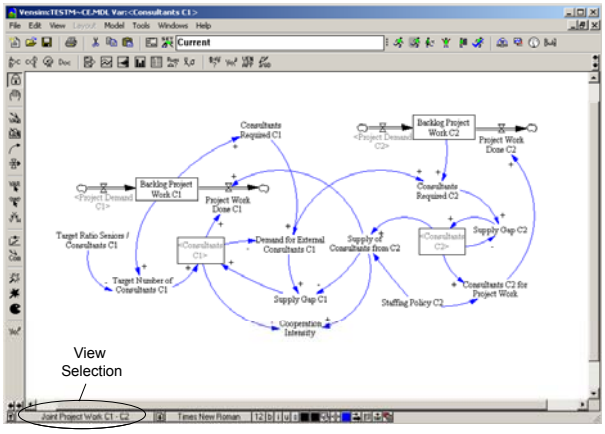

Figure 16: View of “Joint Project Work C1 - C2”

Simulation and analysis of the composed System Dynamics model are done in Vensim. Vensim provides a comprehensive set of tools to analyze the model structure as well as the simulation output. Parameterization allows the setup of different scenarios. Traditionally scenario development is restricted to the structure of the underlying model. The proposed component-based modeling approach facilitates model modification and thereby enhances scenario building to include alternative structures. This aspect is especially important in problem domains that deal with structural changes, such as collaboration management and business networking.

CONCLUSIONS

A component-based modeling approach to the development of System Dynamics models basically has two major advantages. The first advantage is obvious. The reuse of existing model components facilitates the model building process. As modeling effort is reduced, the building process is accelerated. Further more, the quality of developed models can be enhanced. The expertise of subject matter experts is encapsulated in design patterns and model building blocks that can be easily composed to solve problems within a certain domain. The second advantage is that the component-based modeling approach can be used to improve the structural flexibility of System Dynamics models as such. The multiple instantiation of a component allows the handling of redundant structures, thus paving the way for the distinction and explicit modeling of individual elements, a privilege predominantly unknown to the System Dynamics approach.

REFERENCES

Eberlein, R. and J. Hines. 1996. “Molecules for modelers”. In *Proceedings of the International System Dynamics Society*. Cambridge: System Dynamics Society.

Myrtveit, M. 2000. “Object-oriented Extensions to System Dynamics”. In *Proceedings of the International System Dynamics Society*. Bergen: System Dynamics Society.

Sterman, J.D. 2000. *Business Dynamics - Systems Thinking and Modeling for a Complex World*. Boston: McGraw-Hill.

Tignor, W.W. and M. Myrtveit. 2001. “Object-oriented Design Patterns and System Dynamics Components”. In *Proceedings of the International System Dynamics Society*. Atlanta: System Dynamics Society.

Ventana Systems Inc. 2003. *Vensim 5 User’s Guide*. Harvard: Ventana Systems.

Ventana Systems Inc. 2003. *Vensim 5 Reference Manual*. Harvard: Ventana Systems.