# ENHANCING SYSTEM DYNAMICS MODELING USING A COMPONENT-BASED APPROACH

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*

**Abstract:** An approach using component technology for the development of System Dynamics models is introduced. At first a brief introduction of System Dynamics, its modeling perspective 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 an approach for component development and model composition. The approach builds upon a conceptual framework that is basic and independent from any problem or domain specific context. An example depicts the appliance of the approach to a given problem domain and explains the usage of domain specific model design patterns and components. Finally a prototype system to support component-based modeling is described and the advantages that component technology brings to System Dynamics modeling are pointed out.

*Keywords:* System Dynamics, Model Components, Conceptual Framework, Model Composition, Software Tool

## PROBLEM

The use of component-based approaches leads 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 an approach 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 2000]. Therefore, concepts for modeling components and software environments [Powersim 2003] 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 models is outlined. After that the component-based modeling approach and a prototype system supporting the component-based composition of models are described.

## SYSTEM DYNAMICS

System Dynamics is an approach that focuses on the analysis of the behavior of complex technical and socio-economic systems. Systems are seen as an interlocking structure of feedback loops [Forrester 1976]. By using simulation models System Dynamics 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. Figure 1 shows the notation that is usually used to visualize the modeling elements.

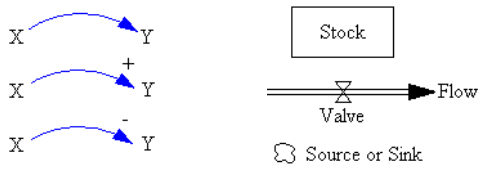



Figure 1: System Dynamics modeling elements

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, p. 139].

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.


Figure 2: Stock and flow

The mathematical representation is:

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

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, p. 194].

By combining these modeling elements the structure of the system to be analyzed is given. Feedback loops are at the core of 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).

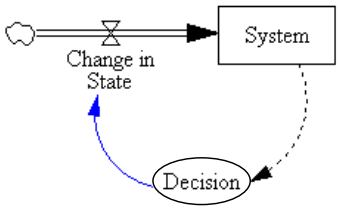



Figure 3: Decision as a feedback process

A decision is made by applying a set of decision rules to the system. The decision is based on information about the state of the system and 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 Perspective**

System dynamics modeling follows a top-down systems view [Bauer et al. 2005]. The high-level structure of the system is sketched providing a conceptualization of aggregate key elements and relationships. Usually the main stocks in the system are identified first, followed by the flows and the relationships that determine the flow rates. During model development initial stocks are gradually decomposed until all relevant feedback loops are captured. System dynamics seeks an endogenous explanation for a given phenomenon based on the identification of dominant feedback structures. Models developed capture emergence by modeling the phenomenon itself [Schieritz and Milling 2003].

Further, a homogeneous perspective is applied to the investigated problem. This is an inherent aspect of the system dynamics approach. System Dynamics models maintain an aggregated view [Scholl 2001]. Individual items flowing through a stock and flow network cannot be handled. Hence the heterogeneity of individual elements and their attributes isn't captured on an individual level as e. g. in agent-based modeling approaches [Bauer et al. 2005].

**Model Representation in Vensim**

Vensim is an interactive software environment for the development, simulation, and exploration of System Dynamics models. 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. Figure 4 shows the structure of a simple System Dynamics model created with the sketch editor.

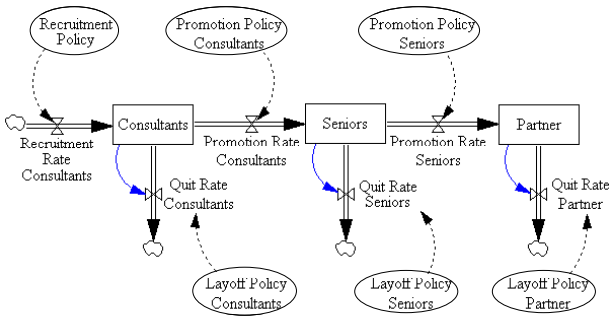



Figure 4: Simplified System Dynamics model capturing the workforce of a consultancy

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 is computed at any point in time t 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 mana-

gement 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.

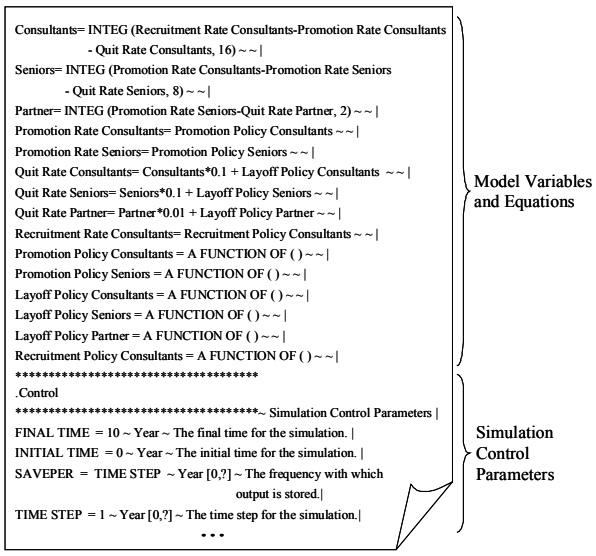



Figure 5: Model variables, equations and simulation control parameters

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 2003, p. 22]. 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 2003, 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.

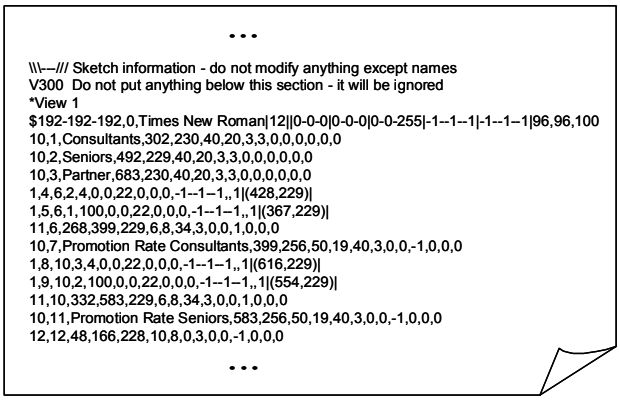



Figure 6: Sketch data

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 2003, p. 383].

## COMPONENT-BASED MODELING

### Conceptual Framework

The component-based approach introduced in this paper builds upon a conceptual framework that defines the elements of the approach and their relationships (see figure 7).

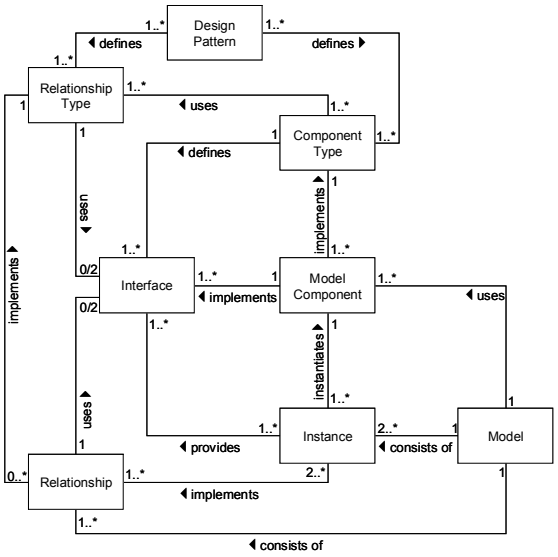



Figure 7: Conceptual Framework

The concept of a model component is central to the framework. A model component is a piece of model that can be used as a building block. A model component corresponds in some way to the concept of a class known from object engineering [OMG 2003]. Thus, a model component is an artifact that encapsulates a well-defined structure. It implements specified interfaces to interact with other model components.

To create a model, components are instantiated, similar to the instantiation of objects from classes. A model consists of at least two instances and a relationship between them. The difference between a model component and an instance is that an instance is at least related to one other instance. The conceptual separation of model components and instances is constitutional for multiple instantiation. The instances of one multiple instantiated component comprise identical model structures. However, implementing different relationships, these instances take unique positions within a composed model.

Relationships between instances are bilateral except for aggregations. A relationship requires complementary interfaces. Two instances are connected by pairing their corresponding interfaces. Aggregations merge two or more instances creating one superordinate instance. To aggregate instances complementary interfaces aren't necessary. The interface of the superordinate instance is simply a collection of aggregated instances' interfaces.

Components with identical interfaces belong to the same component type. A component type is similar to the concept of an abstract class in object engineering [OMG 2003]. An abstract class defines interfaces without implementing them. However, the abstract specification ensures that the instances of components belonging to the same component type can be substituted by each other. Further, interface specification restricts possible relationships as complementary interfaces are necessary. To make these restrictions explicit, relationship types are defined. A relationship type represents an abstract specification of the type and amount of possible relationships between the instances of involved component types.

By defining component and relationship types a meta-structure is created that guides the development of model components, but remains independent of individual implementation. This meta-structure is called a model design pattern. The model design pattern serves the development of component-based models as it describes the attributes and relationships of components to be implemented.

**Component Type Specification**

The superstructure of a model component is defined within a component type specification. A model component contains several model variables, equations and corresponding sketch data. Unlike traditional System Dynamics models, variables of model components are typed. To assign a variable type, the original variable name is extended with a user-defined keyword delimited by the symbol "$". The keyword itself is defined within the component type specification. Figure 8 gives an example.

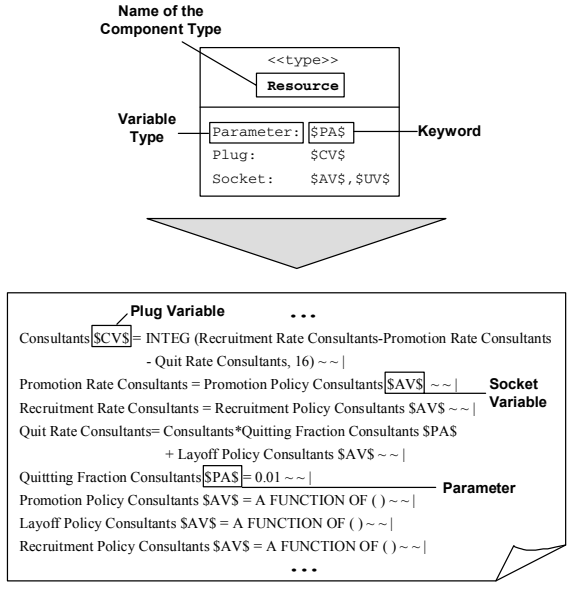



Figure 8: Component Type Specification

Four types of variables are used to specify a component type: parameter, private, plug and socket variables. Parameters take numerical values set by the user. Private variables are part of the model structure that is encapsulated by the component. All variables that are not explicitly marked by a keyword are treated as private variables. Plug and socket variables represent the interfaces of a component. Plug variables connect to socket variables. The component type specification shown in figure 8 defines a component type named "Resource" and specifies parameters to be marked by

"$PA$" while plug variables have to be denoted by "$CV$". For socket variables two variable groups can be distinguished assigning the keywords "$AV$" and "$UV$".

The specification of a component type does not necessarily contain definitions for all of the four variable types. Parameters and private variables are optional. However at least either one plug or one socket variable type has to be defined otherwise connections with other components are impossible.

### Component Instantiation

In System Dynamics models variables are identified solely by name, thus names must be unique. To ensure this uniqueness of names the string "#VAR#" is used as instantiation keyword in every model components variable's name. During component instantiation this keyword is replaced by the instance name of the created entity. Figure 9 gives an example.

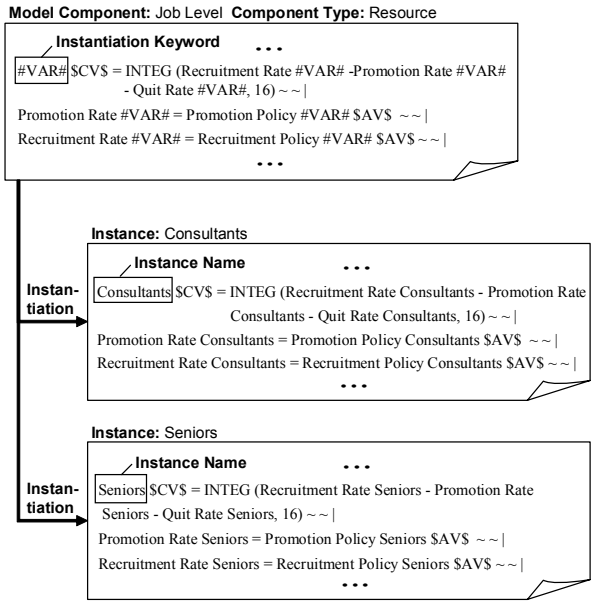



Figure 9: Component Instantiation

The figure shows an excerpt of the content of a model component named "Job Level". The component implements the component type "Resource". The component's variables are marked by the keyword "#VAR#". In addidtion, two instances of this component are depicted. Instances are labeled "Consultants" and "Seniors". Although the structure of both instances is the same, variable names are unique as instance names are distinct. By using the instantiation keyword "#VAR#" as replacement string for the instance name it is possible to create multiple instances of the same component while ensuring that every variable's name remains unique in the composed model.

### Aggregations

Aggregations merge two or more instances of one or more components. The component type of the created superordinate instance is also defined as part of the model design pattern. Figure 10 gives an example.

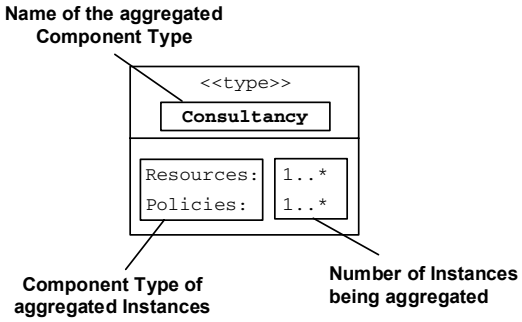



Figure 10: Aggregation

The specification shown in figure 10 defines an aggregated component type labeled "Consultancy". Instead of defining variable types it is specified that "Consultancy" consists of one or more instances of components belonging to the component type "Policy" and one or more instances of components implementing the component type "Resource".

### Relationship Types

The possible set of relationships between components is defined within the design pattern using relationship types. A relationship type specifies the kind of relationship and denominates the component types being involved. Three kinds of relationships are available: aggregate, associate and use.

Aggregate-relationships depict the hierarchical relationship between an aggregated component type and the component types being merged. Figure 11 gives an example.

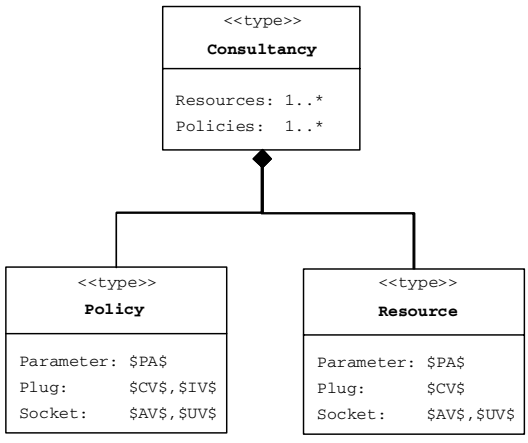



Figure 11: Aggregate-Relationship

The relationship graphically depicts the aggregation that is defined by the component type "Consultancy" in figure 10.

Associate-relationships represent lateral connections between two or more instances. The related instances either stem from one or more components of the same component type or they belong to components implementing different component types. Further an associate-relationship is either one-way or two-way. In a one-way relationship the plug variables of one instance get connected to the socket variables of the other instance. The direction of the relationship is marked by an arrow. In a two-way relationship the plug variables of both instances get connected to the socket variables of the respectively other instance (see figure 12).

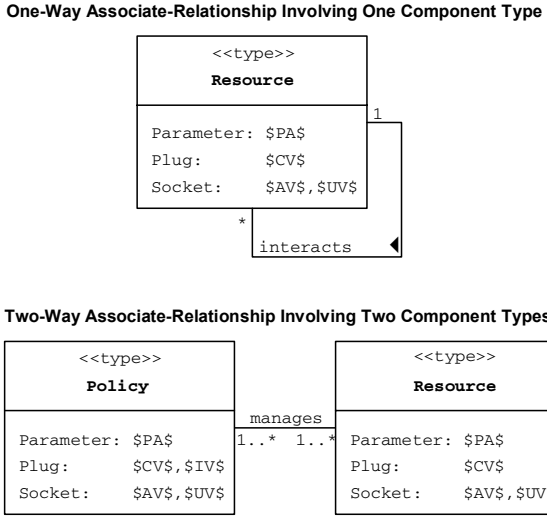



Figure 12: Associate-Relationships

The one-way associate-relationship defines that an instance of a component implementing the type "Resource" can be connected to one or more instances of the same component or to one or more instances of other components that are typed as "Resources". The exemplary two-way associate-relationship defines that an instance of "Resource" must be related to at least one instance of the component type "Policy" and vice versa.

Use-Relationships particularize associate-relationships specifying the groups of plug and socket variables by which a given relationship is implemented. Corresponding to associate-relationships connected instances belong to either one or more components of the same component type or to components of different component types. Use-relationships are always directed. Graphically they are depicted by an arrow with a dashed line. The arrow starts at the component type whose instance's plug variables are used to implement the instance's socket variables of the component type where the arrow ends. The keywords of the paired groups of plug and socket variables are depicted on the arrow beginning with the keyword of the plug variables. Figure 13 gives two examples corresponding to the associate-relationships shown in figure 12.

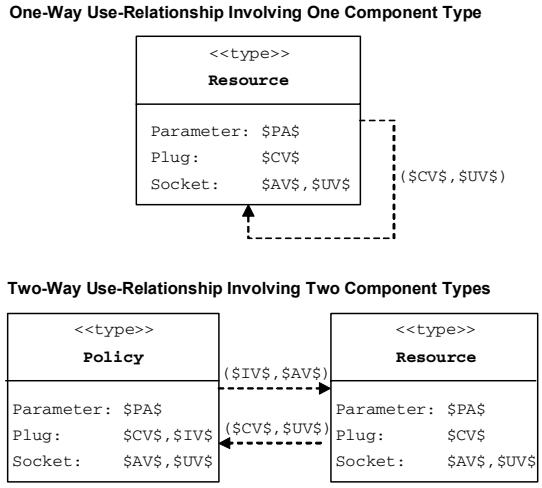



Figure 13: Use-Relationships

The one-way one-component use-relationship defines that instances of the component type "Resource" can be related by connecting plug variables "$CV$" with socket variables "$UV$".

The second example specifies that plug variables "$IV$" belonging to instances of "Policy" connect to socket variables "$AV$" being implemented by instances of "Resource". Plug variables "$CV$" of these instances in turn attach to socket variables "$UV$" comprised by instances of "Policy".

**Composition of Component Instances**

Distinguishing groups of variables the component type specification enables the definition of component interfaces. Together with relationship types, connections between component instances can be modeled on an abstract type level, independent from individual component implementation. As aggregations merely act as containers, they don't need to specify separate plug and socket interfaces to connect to other instances. They draw on the interfaces implemented by the instances they include. Hence, associate-relationships determine together with use-relationships how component instances get connected to form one composed model. However, the specification of associate- and use-relationships only pairs distinct groups of plug- and socket-variables.

To determine which individual plug variable implements a given socket variable, the names of corresponding plug and socket variables have to be identical, except their instance name and variable type keyword. Figure 14 gives an example that illustrates the assembly process.

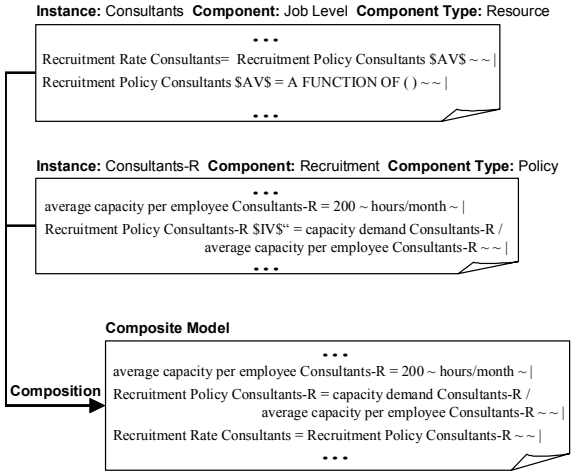



Figure 14: Component composition

Two component instances, "Consultants" and "Consultants-R" are composed. "Consultants" is an instance of model component "Job Level" implementing component type "Resource". "Consultants-R" is an instance of component "Recruitment", whose component type is "Policy". According to the associate-relationship shown in figure 12 instances of "Resource" and "Policy" can be connected. The use-relationships in figure 13 specify that connections are possible either between "Policy"-plugs with keyword "$IV$" and "Resource"-sockets with keyword "$AV$" or between "Resource"-plugs marked by "$CV$" and "Policy"-sockets marked by "$UV$".

In figure 14 instance "Consultants" contains the socket variable "Recruitment Policy Consultants $AV$". This socket variable is used to determine the value of the variable "Recruitment Rate Consultants". 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 ( )". As the socket variable only possesses an equation stub, no value can be computed. Instance "Consultants-R" defines the corresponding plug variable "Recruitment Policy Consultants-R $IV$" and provides an equation to determine the variable's value. During assembly the equation stub of the socket variable is replaced by the equation of the corresponding plug variable. Keywords "$AV$" and "$IV$" are deleted to simplify variable names within the composite model.

### Model Design Pattern

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. Using a component-based modeling approach domain-specific components are used to capture this knowledge. Based on the definition of component and relationship types domain-specific model design patterns can be developed. A domain-specific model design pattern depicts the common meta-structure of models for a given problem domain. It predefines the structure of prospective composite models, thus guiding the model building process. The design pattern describes the attributes and relationships of components to be implemented. In addition, the design pattern can be used to facilitate the process of model composition. Possible relationships between instantiated model components are detected automatically and are suggested to the user.

Figure 15 gives an example of a domain-specific design pattern that depicts the meta-structure of models capturing the collaborative relationships between consultancies.

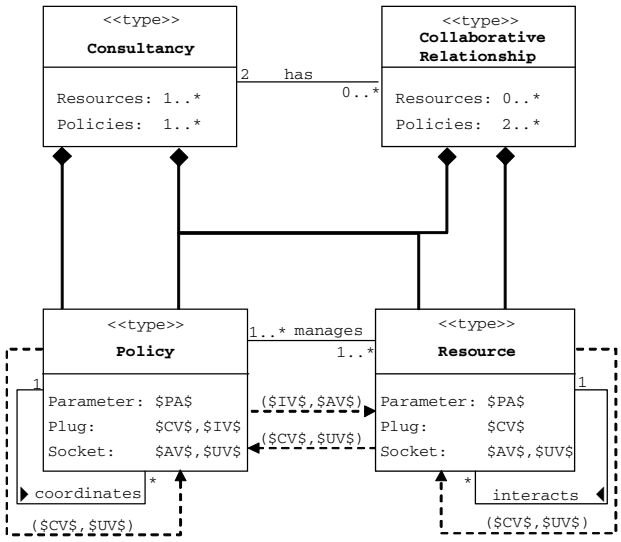



Figure 15: Domain-specific model design pattern

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

As figure 15 depicts, the components of type "Policy" and "Resource" implement the plug and socket interfaces necessary for composition. The relationship "interacts" between resources is implemented by pairs of "$CV$" and "$UV$" variables. The relationship "manages" between a resource and a policy comprises pairs of "$CV$" and "$UV$" variables and "$IV$" and "$AV$"

variables, respectively. As policy instances may also be connected to each other using pairs of "$CV$" and "$UV$" variables, it is possible to assign the government of one policy to another, superior policy instance. 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 16 shows the architecture of the prototype.

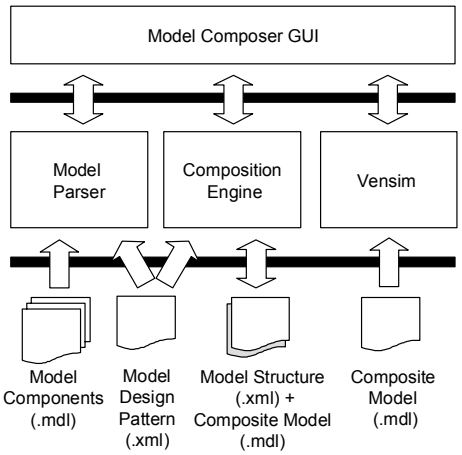



Figure 16: Architecture of the model composer

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 17 shows the graphical user interface of the prototypical model composer.

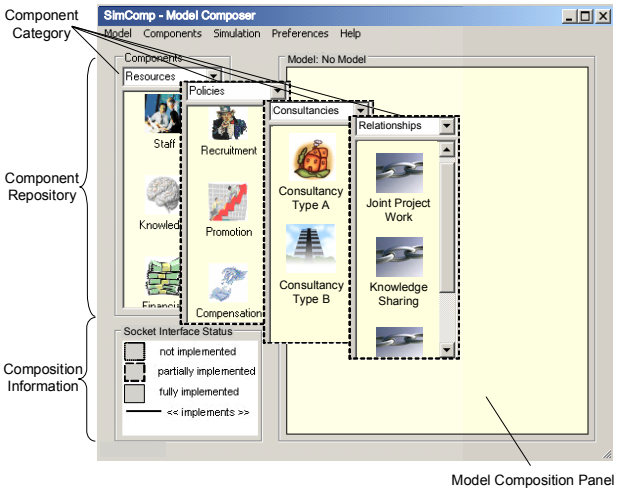



Figure 17: Graphical user interface of the model composer

Available components are categorized according to their component type. As the exemplary model design pattern shown in figure 15 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 18 gives an example of a model being in the composition process.

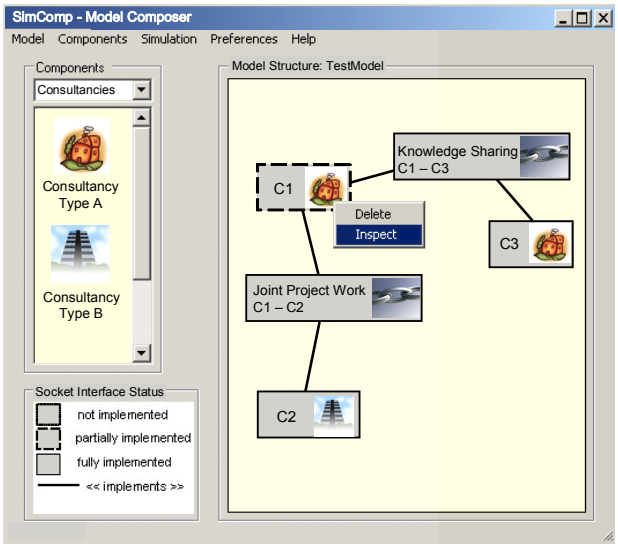



Figure 18: Construction of composite models

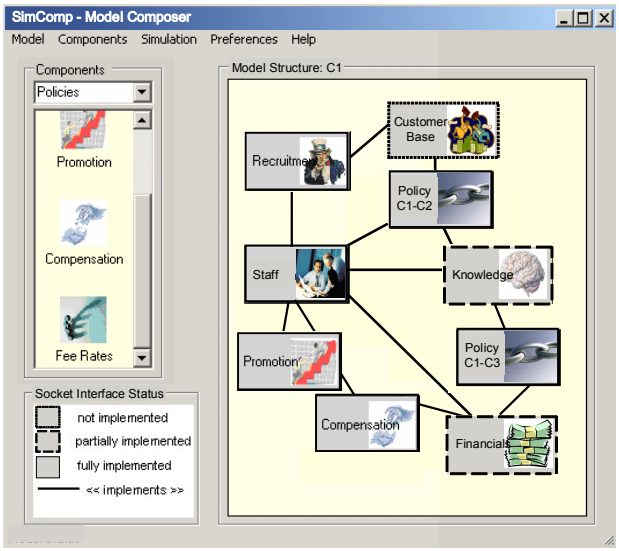



Figure 19: Content of instance "C1"

The model composition panel comprises two instances of the component "Consultancy Type A", one instance of "Consultancy Type B" and two instances of collaborative 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 (see figure 15), the instances shown in figure 18 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 19 shows the content of the 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 19 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 figure 15). 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 keyword. Further the relationships of these variables to corresponding plug, respectively

socket variables of other components are shown. Figure 20 gives an example.

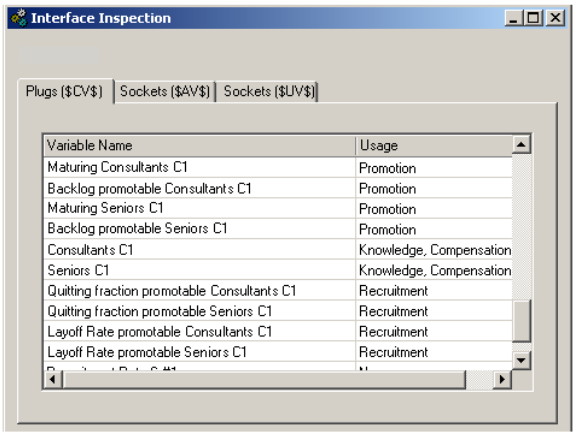



Figure 20: Interface inspection of instance "Staff"

The figure shows the inspection of the instance "Staff". According to the specification of the component type "Resource", the interface variables are defined as plugs ("$CV$") and sockets ("$AV$", "$UV$"). Selecting the tab "Plugs ($CV$) returns a list of all model variables marked by that keyword. The column "Usage" shows, which instances define socket variables that use a certain plug variable. E. g., the variable "Maturing Consultants C1" is used by the instance "Promotion", whereas the variable "Consultants C1" is used by "Knowledge" and "Compensation".

Figure 21 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.

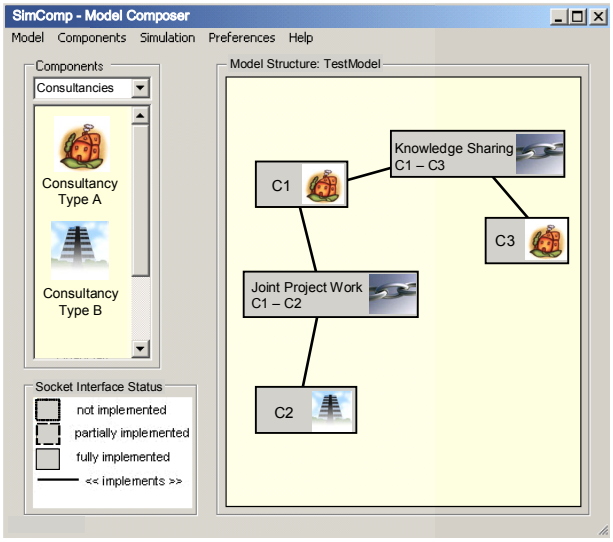



Figure 21: Composed model

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 22 shows the component instance "Joint Project Work C1 - C2" in Vensim.

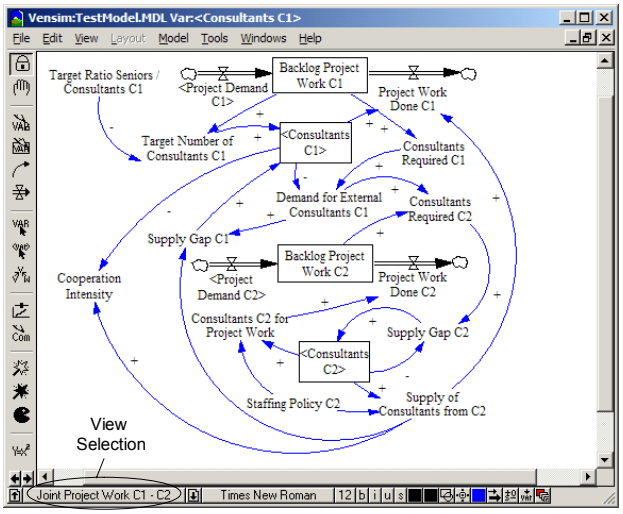



Figure 22: View of "Joint Project Work C1 - C2"

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 [Bauer and Bodendorf 2003].

## CONCLUSIONS

A component-based modeling approach to the development of System Dynamics models 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. In addition, 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 facilitates the distinct modeling of elements with identical or similar structures. Component-based modeling does not revoke the aggregated view inherent to the System Dynamics approach, but it paves the way for the distinction and explicit modeling of individual elements, a privilege predominantly unknown to the System Dynamics approach.

## REFERENCES

Bauer, C., Barbulovic-Nad, L. and Bodendorf, F. 2005, "Comparing System Dynamics and Agent-Based Simulation to Support Strategic Decision-Making". In *Proceedings of the 6th Workshop on Agent-Based Simulation*, Erlangen, ISBN 3936150443, Pp. 56-62.

Bauer, C. and Bodendorf, F. 2004, "Simulating Business Networks in the Consulting Industry with System Dynamics". In *Proceedings of the 2004 Summer Computer Simulation Conference*, San Jose, SCS, ISBN 156552830, Pp. 42-47.

Eberlein, R. and Hines, J. 1996, "Molecules for modelers". In *Proceedings of the International System Dynamics Society*. Cambridge, System Dynamics Society. http://web.mit.edu/jsterman/www/SD96/home.html.

Forrester, J. W. 1976, "Urban Dynamics". Cambridge, MIT Press, ISBN 0262060264.

Myrtveit, M. 2000, "Object-oriented Extensions to System Dynamics". In *Proceedings of the International System Dynamics Society*. Bergen, System Dynamics Society, ISBN 0967291429.

Object Management Group (OMG) 2003, "Unified Modeling Language: Superstructure: Version 2.0: Final Adopted Specification". http://www.omg.org/docs/ptc/03-08-02.pdf.

Powersim Software, AS 2003, "Powersim Studio 2003 User's Guide". Bergen, Powersim Software.

Schieritz, N. and Milling, P. 2003, "Modeling the Forest or Modeling the Trees: A Comparison of System Dynamics and Agent-Based Simulation". In *Proceedings of the 21st International Conference of the System Dynamics Society*. New York, System Dynamics Society, ISBN 0967291496.

Scholl, H. 2001, "Agent-based and System Dynamics Modeling: A Call for Cross Study and Joint Research". In *Proceedings of the 34th Annual Hawaii International Conference on System Sciences*, Vol. 3, Maui, ISBN 0769509819.

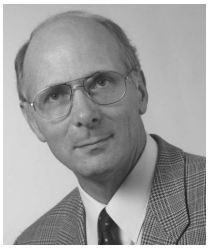
Sterman, J. D. 2000, "Business Dynamics - Systems Thinking and Modeling for a Complex World". Boston: McGraw-Hill, ISBN 0072311355.

Tignor, W. W. and Myrtveit, M. 2001, "Object-oriented Design Patterns and System Dynamics Components". In *Proceedings of the International System Dynamics Society*. Atlanta, System Dynamics Society, ISBN 0967291445.

Ventana Systems, Inc. 2003, "Vensim 5 Reference Manual". Harvard, Ventana Systems.

## BIOGRAPHIES

**Dr. Christian Bauer** was awarded a degree in business administration at the University of Erlangen-Nuremberg in 1999. In 2000, he became a member of the research staff and a Ph.D. student at the Department of Information Systems II at the University of Erlangen-Nuremberg. In 2006 he was awarded a Ph.D in Information Systems. His doctoral thesis is focused on simulation-based systems supporting strategic decisions within business networks in the consulting industry.



**Prof. Dr. Freimut Bodendorf** graduated from the University of Erlangen-Nuremberg (School of Engineering) with a degree in Computer Science. He received his doctor's degree (Ph.D.) in Information Systems. Subsequently he was head of an Information Systems Department at the University of Freiburg, Germany, professor at the Postgraduate School of Engineering in Nuremberg and head of the Chair of Computer Science and Information Systems at the University of Fribourg , Switzerland. Since 1989 he is head of the Chair of Information Systems II at the University of Erlangen-Nuremberg.