Reusability in System Dynamics: Current Approaches and Improvement Opportunities

Ricardo Sotaquirá, Gerly Carolina Ariza Zabala
Systems Thinking Research Group
Universidad Autónoma de Bucaramanga
Calle 48 Nr.39-234 Bucaramanga, COLOMBIA
Tel. +76 43 61 11 ext. 347
E-mail: **rsotaqui@unab.edu.co** - **gariza@unab.edu.co**

http://fis.unab.edu.co/gps/sistemika/index.html

Abstract

Several companies in Colombia, and also some public institutions, are beginning to use formal methodologies for strategic analysis. Unfortunately, System Dynamics modeling is an almost unknown option for these potential users. One answer to this poor diffusion is based on the concept of model reusability. It will be revealed, by means of a framework of levels of reusability derived from the object-oriented approach, that the state of the art in system dynamics reusability shows the same evolution stage of software reusability before the 1960s. This interpretation will expose too a major opportunity for encourage the practice of model reuse in our System Dynamics community and for widening the spectrum of users. A first implementation of a software tool for system dynamics modeling with higher level of reusability is being developed.

Keywords: reusability, model reuse, generic structures, object-oriented approach, inheritance hierarchy

1. Introduction

Several companies in Colombia, and also some public institutions, are beginning to use formal methodologies for strategic analysis. Unfortunately, System Dynamics modeling is an almost unknown option for these potential users. They usually choose a less robust and more qualitative approach like scenario planning or balanced scorecard. Furthermore the local community of system dynamics practitioners is quite small.

This situation is very similar to the one presented by Graham Winch (2002) about the barriers to entry for system dynamics (SD) in small-medium enterprises of industrialized countries. In both cases, a pertinent contribution could be to offer an alternative where "the participation of the consultant/specialist can be partially, or ideally completely, removed" (Winch 2002, p.342). Winch proposes to use a simulator based on a user-parameterized generic model. From a wider perspective, this alternative is just one member of a series of proposals based on the concept of model reusability.

From system archetypes to molecules, a history of model reuse has been written in the system dynamics practice. But this paper will try to prove that the state of model reusability in our field is similar to the state of software reusability before the 1960s, and that we have to pay

close attention to the concept of reusability, as developed by software engineering, in order to achieve a methodological and technological improvement and a wider diffusion of SD.
Therefore, the theoretical construction of the reusability concept in object-oriented software engineering constitutes the first part of the paper[1]. There is no just one approach to software reusability, but there is a series of reusability levels developed during the growing of software engineering field. On the basis of the definition of each level, it will propose a corresponding reusability level in system dynamics. Later, this interpretive framework about reusability will be used to examine an state of the art of system dynamics proposals relative to model reuse: System archetypes, molecules, predefined components, user defined components, etc. Finally, this interpretation will clearly reveal the limitations of the current meaning of reusability in our field, and it will expose too a major opportunity for methodological and technological improvement.

2. Levels of Reusability

As Meyer (1988) pointed out it would be unfair to say that reuse only occurs in software engineering when object-oriented approach arose. Accordingly, it would be unfair to affirm that reuse only occurs in SD until the emergency of system archetypes. But these generic structures constitute one of the first proposals moved by an explicit intention of model reuse. The reusability concept in software engineering appears when the deliberated intention of code reuse was formulated and this event occurs in the domain of the object-oriented approach (OOA).
Reusability concept in OOA has been evolved. From its beginnings on the 1960s to its formal specification several years later, software reuse has been enhanced its power. In order to offer an organized view of this progress we propose a framework composed of four levels of reusability: (1) loosely coupled units of state and behavior, (2) close mapping between components and problem domain or mental model entities, (3) composability/decomposability and white-box encapsulation, and (4) inheritance hierarchy and polymorphism. Current research in software development is still increasing these basic levels with higher ones (i.e. multiple inheritance), but these four levels are shared by current standards in the field.

2.1. First Level of Reusability: Loosely Coupled Units of State and Behavior

Usually the concept of object, or formally defined the concept of class, in OOA is introduced as a "solution" to the gap between data and code, or between state and behavior. This "solution" appears in the context of one of the first computer simulation languages: Simula (Nygaard 1962; Dahl, Myhrhaug and Nygaard 1968). The first implementation of Simula, in 1962, features an innovation in computer programming. The program was organized in independent parts that combine data and code: "customers" and "stations". (Simula was a programming language oriented to queuing models). But in the second version of Simula, they "replaced these [parts] with *one* concept that could describe all the different 'objects' that should participate in a simulation. This concept was called a 'process' and could play both an active [station] and a passive [customer] role in simulations" (Krogdahl 2003, p.2).

[1] The research process presented in this paper is based on an interpretive holistic approach: the Interpretative Systemology (Fuenmayor 1991). Particularly the methodological guidelines of its theory of organizations (Fuenmayor 2001) have been followed.

The version named Simula 67 changed the concept of process by the concept of class. The concept of class, the formal definition of an object, was the cornerstone for the formulation of a software construction methodology based on reusable components, it was the first feature of software reusability.

By definition, system dynamics transcends the gap between state and behavior. However system dynamics modeling generally is not oriented to build models by using independent units. In fact, feedback loops can go through several subsystems or submodels. Therefore one conceivable difference between OOA concept of class and the corresponding concept in SD is that the latter sometimes correspond to *independent* units, but in the general case SD classes correspond to *loosely coupled* units. That is, they would be units with a minimum coupling between each other, just the minimum interdependence to support feedback loops.

2.2. Second Level of Reusability: Close Mapping between Components and Problem Domain or Mental Model Entities

One of the early observed advantages of OOA, with respect to previous software development approaches, was the close relationship between the concept of class/object and the implicit concept of entity that we commonly use to understand our world. This is the reason why OOA practitioners suggest that "object-oriented design is a natural approach: the world being modeled is made of objects … and it is appropriate to organize the model around computer representations of these objects" (Meyer 1988, p.51). In a less positivist fashion, we could affirm that the concept of object allowed to software engineers a one-to-one mapping between the software model and the mental models about the problem domain.

This feature has promoted software reusability. For example, the entire technology in Graphical User Interfaces is supported by a set of common notions about human visual interaction: window, menu, icon, etc. These notions have given to the developers a common set of objects that have been increasingly enhanced with their programming efforts. The one-to-one relationship between software object and real/mental entity encourages collective work around common issues.

In some cases this feature of OOA could be in contradiction with the orientation of SD to underlying structure and its foundation on the concept of feedback information system. Sometimes it could not be possible to draw a one-to-one connection between a class (object or component) in SD and a corresponding mental entity. But to encourage a close, instead of one-to-one, mapping between both domains, through the concept of SD class, means to promote model reusability.

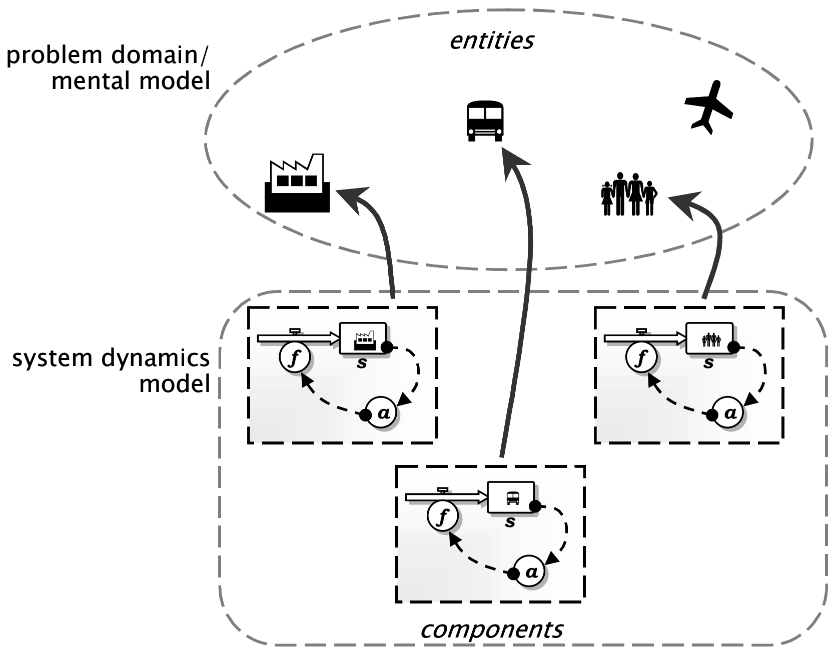

Figure 1. Close mapping between components and problem domain or mental model entities

2.3. Third Level of Reusability: Composability/Decomposability and White-box Encapsulation

The original version of Simula offered to the user two basic predefined classes, station and customer, but it was only until Simula 67 when user-defined classes appear. This event opened a vast land of possibilities for software reuse. The user then could create new classes composed of simple classes previously defined by the programming language or by himself. On the other hand, a developer could easily observe the decomposition of a system into its constitutive components, avoiding extensive changes to the entire system. The formal names of these two OO properties are composability and decomposability, respectively (Meyer 1988, p.12-13).

These object-oriented attributes are complemented by the concept of encapsulation. Any component must have a clear defined boundary that separates its state and its behavior from other components, this is an evident requirement for decomposability. Thus components resemble capsules. In OOA encapsulation is implemented by applying the principle of information hiding, for example in Smalltalk, another object-oriented language of the 1960s, "everything we can describe can be represented by the recursive composition of a single kind of behavioral building block that hides its combination of state and process inside itself and can be dealt with only through the exchange of messages" (Kay 1993).

Information hiding and encapsulation usually imply that components are black boxes. But recently it has been proposed in OOA a white-box approach that also satisfies the requirements of encapsulation and controlled communication between components. The white-box approach would be the appropriate alternative for encapsulation in SD classes and it also fits the Composability/Decomposability conditions.

2.4. Fourth Level of Reusability: Inheritance Hierarchy and Polymorphism

It would seem that an enough level of reusability is achieved when a software technology satisfies the requirements established by the first three levels of reusability. But "classes, as seen so far, are not sufficient. They do provide a good modular decomposition technique. [...]. But more is needed to fully achieve the goals of reusability and extendibility. [...]. Progress in either reusability or extendibility demands that we take advantage of the strong conceptual relations that hold between classes: a class may be an extension, specialization or combination of others. [...]. Inheritance provides this support" (Meyer 1988, p. 217).

Observe that inheritance relationships and composition relationships are different. In the latter ones some classes are integrated into a new and more complex class. This kind of relationship between superclass and subclasses is the relationship between the whole and its compounding parts. On the other hand, if a subclass inherits from a superclass it means that the subclass will extend or specialize the former features of its superclass. Inheritance is a descendant-ancestor kind of relationship. When a programming language supports inheritance then any updates in an ancestor are mirrored automatically in every descendant.

The complement for implementing inheritance is the property of polymorphism. It relates to the ability to replace an instance of an ancestor class by an instance of any of its descendant classes. For example, an object-oriented organizational system has defined a class *operation* and some descendant classes like *sales*, *product design* and *manufacture*. Any reference in the system to an instance of *operation* could be rightly satisfied by instances of *sales* or any other descendant.

Inheritance hierarchy and polymorphism introduced a new dimension of reusability, and they constituted the more distinctive features of reusability in OOA. Classes or components should be sufficiently generic to have a wide domain of use, but also they should offer a simple way to adapt them to a particular use. This dilemma is solved by inheritance and polymorphism.

The application of this level of reusability in SD yields the possibility to have generic structures but, at the same time, to have an efficient technique to fit these structures to particular model circumstances in a quite more flexible and robust way than the approach of parameterizing.

3. Current Approaches to Reusability in System Dynamics

The interpretive framework about reusability formulated in previous section will be used now to examine an state of the art of system dynamics proposals related to model reusability.

All of the examined proposals satisfy at least the first level of reusability, that is the capability of define classes or components. There are other approaches to reusability in SD not oriented to components, for example parameterization of generic models (Winch 2003). These offer a more restricted reusability than any level presented here, so they will not be interpreted in this framework[2]. The following table summarizes the findings of this interpretive study.

[2] Generic models approach also exhibits other shortcomings as Corben et. al. pointed out (1999): a generic model could not be well fitted to the modeled situation, and the group learning process could be hardly limited by the reduction of the modeling process.

Levels of Reusability/ Approaches to Reusability in SD	**System Archetypes**	**Predefined components (i.e. Molecules)**	**Predefined components and limited user defined components (i.e. Liehr)**	**Limited user defined components (i.e. Powersim Studio 2003 and Myrtveit)**
1. Loosely coupled units of state and behavior	Yes. Archetype: generic (abstract) causal loop structure.	Yes. Molecule: generic (abstract) structure made of stocks, flows and auxiliary elements.	Yes. Predefined model components are based on the concept of generic structures.	Yes. A component is a model piece that can be used as a building block of another component.
2. Close mapping between components and problem domain or mental model entities	No. Because an archetype can involve more than one (mental/real) entity, and the archetype is causal-oriented not entity-oriented.	Sometimes the molecule level can map one (mental/real) entity.	Yes. Predefined model components must be designed and named to correspond to entities of mental model.	Powersim does not force that components correspond to mental model entities, the choice depends of the user.
3. Composability /Decomposability and white-box encapsulation	No. Because the boundaries of the archetypes are dissolved when these substructures are integrated into a model.	No. Because the boundaries of the molecules are dissolved when these substructures are integrated into a model.	Yes. The predefined model components can be distinguished into the model and they can be viewed as white boxes depending on the level of aggregation.	Yes. The predefined model components can be distinguished into the model and they can be viewed as white boxes depending on the level in the top-down hierarchy.
4. Inheritance hierarchy and Polymorphism	No.	No. The molecules are organized through a map that pretends to contain inheritance relations. But the implementation does not support ancestor-descendant relationships, there is no inheritance hierarchy.	No. There is composition hierarchy: submodels compose models. This is a hierarchy of different levels of detail or aggregation. Inheritance hierarchy is explicitly not included in this platform.	No. The components are based in the different levels of model abstraction: system, subsystems, and basic structures. This approach does not exhibit inheritance hierarchy, but composition hierarchy.

Table 1. State of the Art in System Dynamics Reusability

The previous table shows that several approaches relate reusability concept with system dynamics. Some of these approaches incorporate only the first levels of reusability, system archetypes (Senge 1990) and molecules (Eberlein and Hines 1996) belong to this stage. In system archetypes, a causal structure cannot be clearly associated with an entity of a mental model because this structure is an abstract representation of a common behavior then it can

simultaneously correspond with different entities. By this reason it could not be achieved a close mapping between mental model entities and causal structures. The same happens to molecules, these substructures can be related to different entities at the same time. In both proposals, when one of these structures is introduced into a model, it loses the possibility to distinguish itself from rest of the model.
Liehr[3] (2002), Myrtveit (2000) and Powersim (2003) have proposed other approaches with a greater level of reusability. In the theoretical proposal of Liehr the predefined components would be generic structures that correspond with mental model entities. In the case of Myrtveit and Powersim the decision to establish the correspondence between mental entities with components depends on the modeler. In both approaches, when this kind of structure is introduced into a model, this can be recognized from the other structures of the model (white-box approach), depending on the level of aggregation. Moreover in Powersim these components can be edited and reformulated on the basis of modeler needs, or new components can be created from more basic components[4].

4. The Elusive Fourth Level of Reusability in System Dynamics

In the following section three proposals of reusability in SD will be closely examined in order to prove that in spite of all of them refer to concepts like inheritance or polymorphism, however the fourth level of reusability is still elusive for current approaches in SD.
Eberlein and Hines (1996) organize their molecules by a diagram. It draws different types of relationships between the molecules. Maybe some of these molecular linkages could be inheritance relationships. But these potential inheritance relations disappear when user is modeling. Because these relationships belong to the diagram, they are not implemented by software. Molecules do not satisfy one previously mentioned requirement of any implementation of inheritance hierarchy, that is, any updates in an ancestor must be mirrored automatically in every descendant during modeling time.
Powersim (2003) exhibits this latter limitation also. The modeler can build a component as an extension of another previously defined. But there is no connection between the former component and the derived one. This is not an ancestor-descendant relationship. On the other hand, the methodological proposal of Myrtveit suggests that "Polymorphism is achieved through the component interfaces, as components with equal interface are interchangeable" (2000, p.1). Later he writes "As an example, a model of a company may have sales channel as a high-level building block" (ibid, p.5). This notion of polymorphism can work if and only if the fundamental concept of inheritance is developed. But this is not the case of Myrveit's paper.
As table 1 shown, both Liehr and Myrtveit/Powersim regard a concept of composition hierarchy, a different kind of hierarchy than inheritance one. Explicitly Liehr's platform moves away from inheritance concept or "principle of heredity" (2002 p.3) as it named.
Arguments presented here confirm that recent approaches to reusability in SD do not satisfy the requirements imposed by the fourth level of reusability: inheritance hierarchy and polymorphism.

[3] Currently the proposal of Liehr is theoretical, that is, there is not a matching software tool.
[4] Barros, Lima and Horta (1991) proposed a similar approach. He defined a limited concept of class with parameterization ability, but neither he did apply the inheritance concept.

5. The Potential Future of Reusability in System Dynamics

The situation of reusability in SD is very similar to the conditions in software reusability before the emergence of Simula language. Simula 67 incorporated all the basic features of reusability described in section 2. So, it is valid to assert that, the state of the art in SD reusability shows the same evolution stage of software reusability before the 1960s, in spite of recent interest in the SD community about this issue.

The availability of a SD modeling technique with inheritance hierarchy and polymorphism would be considered a major instrument for encourage the practice of model reuse in our community of practitioners and for widening the spectrum of users. On one hand, the application of inheritance principles promotes a permanent exercise of generalization-specialization so model reuse and class design could became usual practices in SD field. On the other hand, the ancestor-descendant technique is a lot more powerful way of extending and adapting predefined classes than parameterization technique. The latter one does not allow adding new features to the predefined classes, so its adaptation ability is quite limited. While the opposite occurs with inheritance, even a non expert user would be benefited by having a set of white-box SD classes fully adaptable to their particular modeling needs. Both effects of reusability through inheritance would yield a powerful alternative for SD modeling as an alternative for strategic analysis, specially under the previously mentioned circumstances of Latin-American countries or even those of small-medium enterprises in industrialized nations. It appears to be a pertinent answer for the scarce diffusion of SD approach in the strategic management field.

Finally, a team of our research group is currently developing a first implementation of a software for SD modeling with fourth level of reusability, it is expected to be available in the mid of 2004.

Acknowledgement

This research project has been funded by Instituto Colombiano para el desarrollo de la Ciencia y la Tecnología "Francisco José de Caldas" (COLCIENCIAS), Universidad Autónoma de Bucaramanga (UNAB) y Pensemos Ltda. Our thanks to the Engineering team at Pensemos for their contributions to this project.

References

Barros M.; Lima C.; Horta G. 2001. From Metamodels to Models: Organizing and Reusing DomainKnowledge in System Dynamics Model Development, Proceedings of System Dynamics Conference, Atlanta, Georgia, USA. July. System Dynamics Society.

Corben D.; Stevenson R.; Wolstenholme EF. 1999. Holistic oild field value management: using system dynamics for 'intermediate level' and 'value-based' modelling in the oil industry, Journal of the Operational Research Society, 50: 383-391.

Dahl O; Myhrhaug B; Nygaard K. 1968. SIMULA 67 Common Base Language, Norwegian Computing Center.

Eberlein RJ, Hines JH. 1996. Molecules for Modelers, Proceedings of the 14th International System Dynamics Conference (CD-ROM), Cambridge, Massachusetts, USA. July. System Dynamics Society.

Fuenmayor R.L. 1991. Truth and Openness: An Epistemology for Interpretive Systemology, Systems Practice, 4(5): 473-490.

Fuenmayor, R. 2001. Interpretando Organizaciones, 1st edn. Universidad de los Andes (Venezuela).

Kay A. 1993. The Early History of Smalltalk, ACM SIGPLAN Notices, 28(3): 69-95, ACM.

Krogdahl S. 2003. The Birth of Simula, Proccedings of the HiNC 1 Conference, Trondheim, June. IFIP.

Liehr M. 2002. A Platform for Systems Dynamics Modeling: Methodologies for the Use of Predefined model Components, Proceedings of the 20th International System Dynamics Conference(CD-ROM), Palermo, Italy. July. Systems Dynamics Society.

Meyer B. 1988. Object-Oriented Software Construction, 1st edn. Prentice Hall International (UK) Ltd.

Myrtveit M. 2000. Object Oriented Extensions to System Dynamics, Proceedings of the18th International System Dynamics Conference(CD_ROM), Bergen, Norway, August. System Dynamics Society.

Nygaard K. 1962. SIMULA: An Extension of ALGOL to the Description of Discrete-Event Networks, Proceedings of the IFIP congress 62, Munich, North-Holland Publ.

Powersim. 2003. Powersim Studio Express 2003 Online Help. Powersim AS.

Senge PM. 1990. The Fifth Discipline. Doubleday.

Winch G. 2002. User-parameterised generic models: a solution to the conundrum of modelling access for SMEs?, System Dynamics Review, 18(3): 339-357.

Back to the Top