Complementing System Dynamics with Object-Role Modeling

P. (Fiona) Tulinayo[1], Andreas Größler[2], S.J.B.A. (Stijn) Hoppenbrouwers[1],
P. (Patrick) van Bommel[1]

[1]Computer Science Department
Faculty of Science Radboud University Nijmegen
Heijendaalseweg 135
PO Box 9020
6525 GL Nijmegen
The Netherlands
F.Tulinayo@science.ru.nl, stijnh@cs.ru.nl, p.vanbommel@cs.ru.nl

[2]Institute for Management Research
Radboud University Nijmegen
PO Box 9108
6500 HK Nijmegen
The Netherlands
Tel +31 24 3616287, Fax +31 24 3611933
groessler@fm.ru.nl

Abstract:

In this paper we use Object-Role Modeling (ORM) to complement System Dynamics (SD). The art of SD modeling lies in discovering and representing feedback processes and other elements that determine the dynamics of the system. However, SD shows a lack of instruments for discovering and expressing precise, language-based concepts in domains. At the same time, the field of conceptual modeling has long since focused on deriving models from natural expressions. We therefore, turn to ORM as a prime example to complement a strong natural language based conceptual modeling approach into the creation of SD models. ORM is a formal fact-oriented approach for modeling information at a conceptual level. In this paper we investigate the basic building blocks of these methods using examples. Investigating the foundation of the two methods helps us to better understand their underlying concepts and their differences in update behavior due to state and decision changes. We use SD to capture the dynamic, and ORM to capture the static aspect of a system.

Keywords*: System Dynamics, Object-Role Modeling*

Introduction

The two methods we discuss in this paper differ in a number of ways but we use them to capture the static (ORM) and dynamic (SD) aspects of a system. SD as a method has its focus on capturing the structure and behavior of systems composed of interacting feedback loops. ORM aims at modeling static objects, and pictures the world in terms of *objects* that play *roles.* It includes procedures for mapping between conceptual and logical levels and was originally conceived for data modeling purposes (Halpin 1998). To understand the structure of a system, its domain needs to be well understood. This is why we introduce static ORM to enable modelers to properly and systematically conceptualize the domain. ORM is comparable to Entity Relationship (ER) Diagrams in use (Chen 1976). It is, however, a *fact-oriented* approach for modeling information and querying the information content of a business domain at a conceptual level (Halpin 2003). Fact-orientation means that it includes both types and instances in its models; types are called “fact types”, instances “facts”. Including the instance level is crucial in linking concepts with advanced SD modeling. We also use ORM because of its strong verbalization and conceptualization facility and for its fully formal link to predicate logic. ORM has a graphical constraint notation that is claimed to be far more expressive for data modeling purposes than, for example, Unified Modeling Language (UML) class diagrams or industrial Entity Relationships (ER) diagrams (Halpin and Wagner 2003). Basically, conceptual modeling is concerned with identifying, analyzing and describing the essential concepts and constraints of a domain with the help of a modeling language (Guizzardi et al., 2002). However, Halpin and Wagner do state that *`although ORM supports modeling of business terms facts, and many static integrity constraints and derivation rules, it cannot model the reactive behavior of systems which can be described using dynamic integrity constraints''*. Clearly we use SD to capture the reactive behavior of a system. Sharif (2005) further states that; ``*....there is a strong case for starting to apply systems dynamics methods more openly in the BPM and MIS research fields, as I feel the tools and techniques available are vastly under-rated in terms of their applicability and capability to provide novel representations of real-world situations.....''*. These complementary statements are the basis for our overall research direction: Complementing System Dynamics with Conceptual and Process Modeling.

Compared with the concepts of ORM, whose roots can be traced back to the 1970’s, the scientific concept of feedback (Richardson 1991), which is at the core of system dynamics modeling, is significantly older. SD is well known for improving the understanding of complex dynamic systems (Bollen et al., 2002). There have been earlier comparative studies, concerning methods potentially complementary to SD, for example SD and Discrete-event system (Brailsford and Hilton 2000; Borshchev and Filippov 2004), and SD and Petri nets (Duggan 2006). In these comparisons the main differences between SD and these methods have been highlighted. In 1996, Richardson identified a number of issues for the future of system dynamics one of which was “understanding model behavior”. To understand the behavior of a model one needs to understand its domain, the connections between the model structure, and behavior in a

sequential manner (i.e. from simpler models to more complex models). This explains why in this paper we present the basic concepts for investigations in the behavior of the two methods as a basis for carrying out more complex modeling.

We carry out this study because it is hard to define complex dynamic models in complex organizational settings therefore, we need support based on ontology (conceptual structure). Secondly, for transferability purposes that's incases where information from one organization need to be reused by another. Lastly to be able to have a basis for the development of a tool that will aid in understanding model behavior.
To achieve this we will take a stepwise approach that's: Identifying the conceptual link between SD and ORM and later with Petri nets (Tulinayo et al., 2008), Identify the key concepts as used in different methods, Map their constructs, Derive transformations (Tulinayo et al., 2009), Create their syntax and semantics, and develop requirements specifications on which a tool can be based.

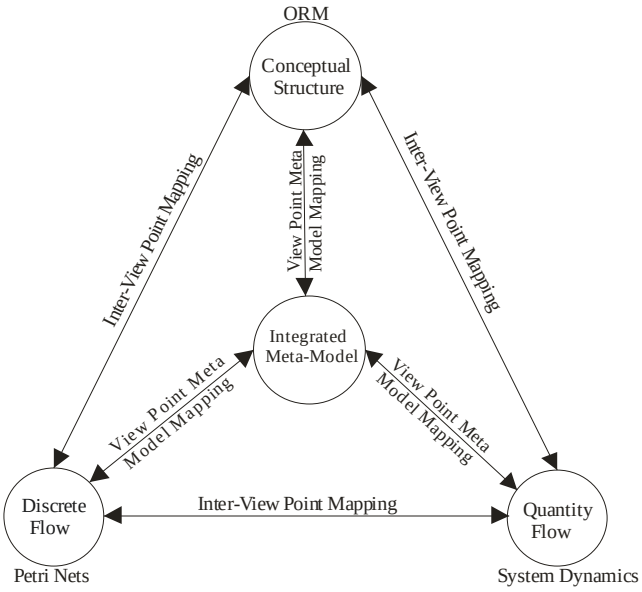

Figure-1: Abstract View of the overall structure

We include Fig-1 to depict an abstract view of the overall structure of what our longer term project is (of which this paper is only part). Two types of mappings are shown: mappings between viewpoints are what we refer to as inter-viewpoint mappings and the mappings between specific viewpoints and integrated meta-model are refereed to as viewpoint meta-model mappings. This project intends to entail the use of three different methods (SD, ORM and Petri nets[1]) to develop a tool that will enable stakeholders (simulation modelers, users and domain experts) to view and manipulate/interact with the integrated models on a common platform. In

[1] A Petri net (also known as a place/transition net or P/T net) is one of several mathematical modeling languages for the description of discrete distributed systems. A Petri net is a directed bipartite graph, in which the nodes represent transitions (i.e. discrete events that may occur) and places (i.e. conditions).

the current study we only use ORM, as a graphical representation for conceptual structure. Petri nets will be introduced later to model a discrete flow, and SD to model a quantitative flow. We use these methods because ORM adds high quality formal conceptualization to SD modeling; Petri Nets will serve to bridge the gap between static ORM and Dynamic, flow-like aspects of SD. We note that both Petri nets and SD are dynamic methods and have commonalities between them, unlike ORM that deals with static conceptualization of the system.

The structure of this paper is as follows. In section 2 we give a summary of the results of earlier papers (Tulinayo et al., 2008 and Tulinayo et al., 2009) where we explored the conceptual link between the two methods, identified commonly used variables, and mapped their constructs. In section 3 we investigate the basic building blocks of these methods using examples. Investigating the foundation of the two methods helps us to better understand their underlying concepts and their differences in update behavior due to state and decision changes. By so doing, we will have a clear guide as we construct more complex models in the future for good foundations giving strong and lasting constructions.

2. Mappings between Methods

In this section we start by identifying the key variables (elements) in ORM and SD. We try to make explicit the relationships between the two methods as illustrated in (Tulinayo et al., 2009). This helps us in charting and comparing the different concepts as used in the methods. Apart from identifying the connections or justifiable similarities between the two methods, we also note the transitional statements against each pair of concepts. The similarities concern ways in which the concepts interact amongst themselves, or the roles they play in the process of modeling systems. First we illustrate some of the basic underlying concepts of the two methods, on which we base our study.

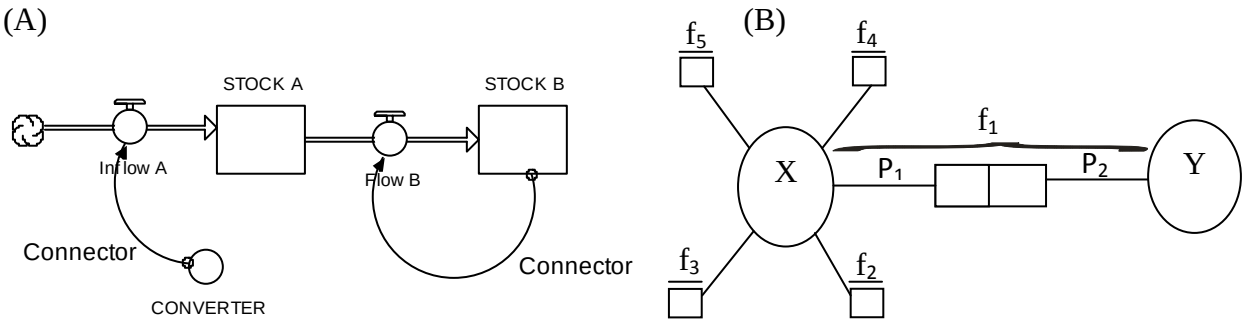

Figure-2: Informal example of SD and ORM

Fig-2 depicts the key concepts that we use/ discuss in this paper; System Dynamics under (A) and Object Role modeling under (B). In SD we use *Stocks* (Stock A and Stock B). These act as reservoirs containing quantities describing the state of the system. *Flow* rates (Inflow into stock A and Outflow from Stock A into stock B) can be imagined as pipelines with a valve that controls the rate of accumulation in the stocks. *Exogenous variables* (Converters) feed new information into the flow rate causing it to change. *Information links* (Connectors) are defined as links that relay information (in the vector sense) from a converter and stock into or out of flows.

In ORM we use *object types* (X, Y) to denote entities (at type level) and *fact types* (f_1, $f_{2, ..., } f_5$) as predicate-like relationships between object types. Object types are a collection of objects with similar properties, in the set-theoretical sense. *Fact types* represent associations between object types, consisting of a number of roles denoting the way object types participate in that fact type. We can have, for example, *unary fact types* ($f_{2,}$ $f_{3,}$ $f_{4,}$ f_5) and *binary fact types* (f_1). Semantically fact types correspond to predicates in predicate logic.

In Table.1 we identify the commonalities or relationship that the basic concepts have among them comprising of SD with ORM plus their transitional statements (a way to tie perceptions in different areas and modeling techniques together) and elements.

System Dynamics	**ORM**	**Transitional Statement**	**Elements**
Stock	Unary fact types	They all contain “things”.	Containers
Quantity	Objects	These can be looked at as the things that flow within the system.	Contents
Flows (Inflow and Outflow)	Object types	They all connect items: Stocks (SD), Unary fact types (ORM).	Homogeneous connectors
Connectors	Fact types	They are all active and involve activities that cause a change to the recipient / destination.	Heterogeneous connectors

Table 1: SD, ORM plus transitional statements

In Table-1 we find connections between different concepts used in the methods. Stocks in SD are similar (though not identical) to unary fact types in ORM because they both contain ``things'' and

we call their elements containers because of their purpose. Quantity in SD is similar to counting Objects (i.e. instances of objects types) in ORM. This is because we look at them as quantities that flow within the system or process. We have the Quantities (SD) and Objects (ORM) which we consider to be related because they all flow within the system/process and their elements are contents. We use the term ``quantity" in SD to represent the items or quantifications that flow within the system. Next we link flows (inflows and out flows) in SD to Object types. This is because they connect different stocks (SD) and Unary fact types (ORM). Hence, we refer to their elements as homogeneous connectors because they all connect similar concepts. Finally we link connectors with fact types because they are both active and have activities involved that cause a change to the recipient/ destination hence their elements being heterogeneous connectors.

3. Basic Concepts for ORM-SD investigation

3.1 Foundations

Richardson (1996) states that *"understanding connections between model structure and behavior comes from a sequential modeling process that is from simpler formulations to complex structures"*. That is why in this paper we start by presenting the basic building blocks as a basis on which to build more complex models later.

In ORM semantic connections between entity types are depicted as combinations of boxes and are called *fact types.* Each box represents a role and must be connected to an *entity type*, a *value type* or *a nested object type*. A fact type can consist of one or more roles. The number of roles in a fact type is called fact type arity. The semantics of the fact type are put in the *fact predicate.* More details of ORM symbols can be found in (Halpin 1998) where he explains their exact meaning.

Figure-3: Basic building blocks

Fig-3 depicts the basic building blocks of SD and ORM. Fig-3A is the ORM representation where *objects in A* play a *role Y (unary fact type).* The population instances of *object type A* are refered to as *objects*. When this is applied to SD we get Fig-3B where *inflow A* is the depiction of when an instance starts to play a role and *outflow B* represents when that instance stops to play that role. The instances that exist within *stock X* are what we refer to as quantities. The quantity of a stock is equal to the number of instances of *object type A* playing role *Y*

3.2 Differences in update behavior

There are significant differences in the way populations of fact types are updated which we refer to as update behavior. In ORM the roles are ordered, which corresponds to the *inflow-outflow* in SD where quantities flow into a stock and then out of the stock on completion of a task/role. The inflow represents an action of starting to play a role and an outflow action of stopping to play that role. In the examples below we focus on the decision whether instances of object type person play that role or not.

3.2.1 Single state change with a single decision

The single state change with single decision occurs when an event is triggered once and never again. For example: *Person was born in EU (single state change with a single decision)* A person can only be born once therefore the decision needs to be made once when he/she is born.

Figure-4: Single state change with single decision

In the ORM model (Fig-4A) we have *person* as the object type and *born-in-EU* as the Unary fact type. The object type *person* plays a role *born-in-EU.* If the person is not born in EU then he does not play that role. In the SD model we have a converter/exogenous variable feeding new information into the inflow (*EU birth*) which is activated every time a person plays the role *born-in-EU*. This will cause a change to the stock (*born in EU)* as represented in Fig-4 B. In this case the SD model will not have an outflow because once a person starts to play that role the state never changes.

3.2.2 Single state change with multiple decisions

A single state change with multiple decisions exists when there is an occurrence of the event that is likely to appear again in the system. For example; *person has visited EU.* Once a person starts to play the role *has-visit-EU* he never stops. This person may decide to play this role again and when he does play this role again after a period of time, multiple decisions are made. This leads to updates in existing information hence, the latest visits and frequency of the visitor are captured. In Fig.5B there is no outflow because once a person starts to play that role he never stops.

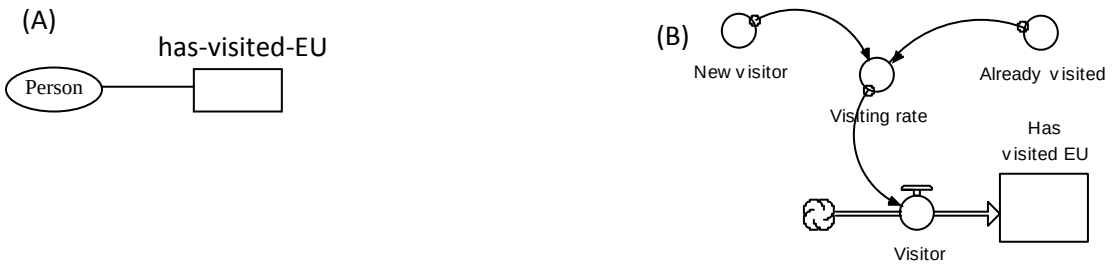

Figure-5: Single state change with multiple decisions

In the single state change with multiple decisions, the ORM model will not differ from the one in Fig-4A although the information changes; yet for the SD model there is a significant change because of the multiple decisions made as reflected by the converters and connects. The converters (new visitor and has visited) represent the new information that is to be fed into the inflow (visitor) through the visiting rate, causing a change to has visited EU.

3.2.3 Multiple state changes

For multiple state changes, a number of state and decision changes occur, for example: *person Lives in EU*. A person can decide to live in EU for a period of time and then get out of EU or decide to move back and forth if need be.

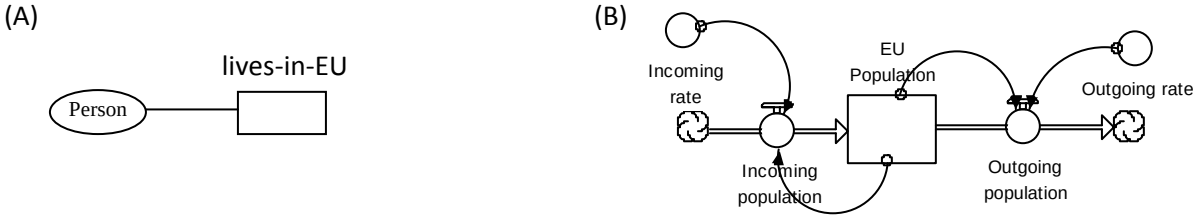

Figure-6: Multiple state changes

The ORM model will not differ from the previous models because object type *person* will play the role *lives-in-EU*, but for the SD model there is a difference because it has to capture all the decisions made. There are information links from the stock (EU population) indicating that EU population influences both the incoming and outgoing population. Here we notice that the SD model has an outflow which reflects the population moving out of EU. The SD model captures this information which enables analysis of the total EU population and the rate at which persons move in and out of EU. For the three given examples we notice that there is no difference in the way the ORM model appears, yet in the SD model there are changes in appearance depending on the decisions made.

In summary, in Fig.4-6 we show the difference in SD and ORM model update behavior. We notice that through all the three examples the ORM model does not change. This is because it does not capture the decisions as they occur, instead it captures the instances of object types as

playing a role (which shows its static aspect) In SD all decisions made are captured (dynamic aspect). Hence, the SD model change depends on the number of decisions made.

Conclusion and further work

The ORM methodology equips the modeler with strong conceptualization of the domain, which is key in developing any model. In this paper we have identified the basic building blocks of both methods, enabling us to understand how these methods differ in their update behavior as a result of state and decision changes. This will enable us to devise better clarifications as we build more complex models in the future.

This research is part of a larger project aiming to improve SD modeling by deploying methods and techniques from system development. We expect that the SD models produced this way will be better understood, with fewer errors, than is currently the case. This will have to be empirically confirmed. With higher quality SD models in place, decision makers and stakeholders should, for example, be able to make better decisions concerning their enterprise and its processes. We will apply the approach presented above in the context of various case domains within the discipline of enterprise engineering. We will further develop and refine the method (its models as well as the stepwise modeling process), by devoting more attention to formalizing its syntax and semantics, but also to the operationalization of the modeling procedures. In addition, we intend to use the techniques suggested in this paper in collaborative settings (Group Model Building; Rouwette and Hoppenbrouwers, 2008), which is a sub-discipline within the field of SD. Finally, we intend to explore further links between SD and process modeling (already initiated by the Petri net involvement), in particular with the YAWL method (Aalst and Hofstede, 2005).

References

1. Aalst, W.M.P.v.d., Hofstede, A.H.M.t. (2005) YAWL: Yet Another Workflow Language. Information Systems, 30(4), 245-275
2. Bollen, L., Hoppe, H.U., Milrad, M., Pinkwart, N. (2002). Collaborative Modelling in Group Learning Environments. In Davidsen, Mollona, Diker, Langer & Rowe (eds), Proc. Of System Dynamics Society, Palermo (Italy), July 2002,
3. Borshchev. A and Filippov.A. (2004) From System Dynamics and Discrete Event to Practical Agent Based Modeling: Reasons, Techniques, Tools. The 22nd International Conference of the System Dynamics Society. Oxford, England
4. Brailsford, S.C. and Hilton, N.A. (2000) A Comparison of Discrete Event Simulation and System Dynamics for Modeling Healthcare Systems. Proceedings of ORAHS, Glasgow Caledonian University, pp. 18-39

5. Chen, P.P. (1976). The entity-Relationship model-Towards a unified view data. ACM Transactions of database systems 1(1), 9-36
6. Duggan, J. (2006) A Comparison of Petri Net and System Dynamics Approaches for Modelling Dynamic Feedback Systems. 24th International Conference of the Systems Dynamics Society, Nijmegen, The Netherlands, July
7. Forrester, J.W. (1961) Industrial Dynamics. The MIT Press, Cambridge
8. Guizzardi, G., Pires, L. F., Sinderen, M. J. v. (2002) On the role of Domain Ontologies in the design of Domain-Specific Visual Modeling Languages. In: Proc. 17th ACM Conference on Object-Oriented Programming, Systems, Languages and Applications OOPSLA
9. Halpin, T. (1998) Object-Role Modeling (ORM/NIAM), Handbook on Architectures of Information Systems, Springer, Heidelberg, Ch. 4
10. Halpin. T and wagner G. (2003) Modeling reactive Behavior in OR. LNCS 2813, 567--569. Springer-Verlag Berlin Heidelberg
11. Richardson G. (1996) Problems for the future of system dynamics. System Dynamics Review 12: 141-157
12. Richardson G. P. (1991) Feedback thought in social science and systems theory. Philadelphia: University of Pennsylvania Press.
13. Rouwette, E., Hoppenbrouwers, S.J.B.A. (2008) Collaborative systems modeling and group model building: a useful combination? 26th International Conference of the System Dynamics Society
14. Sharif, A. M. (2005) 'Industrial Viewpoint' can systems dynamics be effective in modeling dynamic business systems? Business Process Management Journal 11(5): 612--615 q Emerald Group Publishing Limited
15. Tulinayo, P.F, Hoppenbrouwers, S.J.B.A., and Proper, H.A.E. (2008) Integrating System Dynamics with Object-Role Modeling. IFIP International Federation for Information Processing. J. Stirna and A. Persson (Eds.): PoEM, LNBIP 15: 77-85
16. Tulinayo, P.F, Hoppenbrouwers, S.J.B.A., van Bommel Patrick and Proper, H.A.E. (2009) Integrating System Dynamics with Object-Role Modeling and Petri nets. Technical paper, ICIS, Radboud University Nijmegen.