UML and System Dynamics Modeling

Dr. Heinz Schild

Management Consultant and Senior Researcher

with the Gyosei Institute of Management Studies in Reading/UK

Hopmannstrasse 6

D-53177 Bonn

Germany

+49 228 32 37 02

Heinz.Schild@Dr-Schild-Consultants.com

Keywords: Deployment Problems, UML, Use-Case Approach, Object Diagrams, Specific System Dynamics Stereotypes, Recommendations

Deployment Problems

System dynamics models are only rarely used to solve practical problems in commercial enterprises and public administrations, at least in the German-speaking world, probably also in all other industrialized countries. This is surprising considering for how long this approach has been known, not least thanks to the Club of Rome models, and taught at universities. Furthermore, it is available in the form of several powerful and inexpensive software packages. The fact that it is rarely put to practical use is regrettable, for many faulty plans and wrong decisions could demonstrably have been avoided with the help of system dynamics models, as case studies have shown. This situation would seem to arise from two causes. One has to do with the development of mental models in general and the other with the way in which system dynamics problems are tackled.

Mental Models

The difficulty of expressing mental models in formal language – always hard to resolve – has been exacerbated in the last decade by the widespread use of spreadsheet software. The paradigm underlying a spreadsheet is applied to other questions too frequently and too hastily, without determining whether it is suitable for the matter in hand. The resulting models are mainly confined to definition equations. Explicit behavioural equations, if they occur, tend to avoid more complex dynamic interdependencies and feedbacks because of the limitations inherent in a spreadsheet, thus not accounting accurately for the really interesting aspects of a problem.

This can be avoided to a certain extent if while working on a problem a "language" or software tool is used which allows the formal description of mental models without at the same time determining the implementation method.

Organization of problem solving

The second cause is related to the way in which new tasks are tackled in organizations. In system dynamics case studies one often comes across the "heroic" model builder [i]: a manager has recognized a problem, outlines it to one of his colleagues or a consultant who happens to be available and has the relevant experience, who will then overnight deliver a functioning prototype of a reasonably useful system dynamics model. The model then need only be refined and made consistent, which can however take several weeks or months. In reality, several or even many people will normally have to progress through a complex procedure before a task has been sufficiently specified, detailed and structured to determine which of its aspects lend themselves to a formal model or maybe even to a system dynamics model. Even when this has been ascertained, domain experts will still have to give a textual and diagrammatic depiction of the organization's environment in such a way that the mental model required for system dynamics modeling can be derived.

Unified Modeling Language

Some years ago Unified Modeling Language (UML) [ii] was standardized by combining several methodologies all concerned with the processes and tools involved in software development. By now there are computer systems which facilitate working with UML or are based on it. The body of people who understand the concept and are well versed in its artefacts is being widened daily by including UML in the curricula of all relevant educational and training establishments. In addition, it is further promoted through relevant publications and software tool vendors.

The paper will now discuss to what extent UML could overcome or at least minimize the two problems outlined above in deploying system dynamics models more widely.

In the early stages of a project UML primarily aids the communication between the people and organizational units involved. Only at a later stage is it used to formulate requirements and implementation specifications. UML includes a number of precisely defined symbols and conventions regarding their use, based on the paradigm of object-oriented programming, but it does not define the project process. Software tools [iii] based on UML often include basic process recommendations. A common feature of these tools is to take so-called use cases or use-case diagrams as a starting point. The first step is to give a full picture of the application domain by means of use cases.

The Use-Case Approach

A use case describes a set of activities from the point of view of their actors, providing a result which can be observed by at least one of the actors. A use case is always initiated by an actor. In a use-case diagram there will be a symbol for the set of activities and further symbols for the various kinds of actors. Representing all use cases in one diagram will delineate the outer borders of the domain; this is extremely helpful when forming a consensus among the people involved.

Besides graphical representation of use cases, UML tools provide for textual representation in the form of standardized outlines; in this way it is possible to formulate preconditions, constraints, rules or invariants. "Include" and "extend" relationships between the use cases can be introduced in the next stage, and other use cases broken down into sub-use cases. There are symbols and textual descriptions for the actors too. Thus one can distinguish between internal and external actors. Sometimes a system, such as an IT or a telecom system, will be an actor. In UML, different object types, called stereotypes, can be represented using either different symbols or standard object symbols containing a stereotype definition placed in pointed brackets within the symbol.

Object Diagrams

After the first version of the use-case diagrams has been completed the original goals of the project should be redefined and the problems which are to be addressed should be discussed with all those involved in the project and prioritized. Here the visualizations provided by the UML artefacts are very useful for communication and documentation purposes.

In a next step the use cases must be broken down into objects, and the relationships between them and the actors visualized in UML diagrams. Using predefined stereotypes or project-specific ones, the various object types, for instance "level" and "rate" objects, can be depicted. UML offers various ways of representing the relationships between objects, depending on whether they are static or dynamic. Furthermore, several well-defined diagram types are available to illustrate the type of relationship.

UML is an excellent tool for generating collections of generic business patterns [iv] which can be included in other projects. This is a step towards the use of archetypes, as practised in system dynamics.

Developing a UML diagram evolves in several, sometimes iterative, work stages in work groups of differing composition. Co-ordination and communication as well as identification of errors and integrity safeguards are made easier through common use of UML and joint access for all project

members to all diagrams. All the diagrams so far with their objects, relationships, descriptions and constraints constitute the mental model of the current problem.

Components

This phase of the project finishes by drawing together the objects into one or, more usually, several components ready to be implemented. Implementation here is to be taken in its broadest sense. It can involve developing a system dynamics model, a data base or a machine or setting up a work flow or a training course. It need not be decided until this stage of the project which method is to be used in implementing a component because UML is independent of methods. When choosing an implementation method, as is now necessary, important factors to consider will be whether the method is suitable for meeting requirements and whether know-how is available or procurable.

Adaptions

If it has been decided that a component should be implemented as a system dynamics model, its UML diagrams may need an overhaul to accommodate the conventions of the system dynamics package to be used. In practice it is has proved useful to include specific system dynamics stereotypes in the object and relationship symbols. Of all the relationships possible in UML, only dependencies are relevant to system dynamics models. Dynamic associative relationships have to be converted into dependencies. The treatment of aggregate and composite relationships will depend on whether the software allows array variables or not. If array variables are allowed, objects on the aggregate or composite level will often be sufficient. In order to stress the feedback loops so important in system dynamics models sequence diagrams supplemented by constraints can be used.

Recommendations

Unfortunately no system dynamics software package familiar to me [v] allows developers to use UML diagrams instead of proprietary interfaces. UML is certainly rich enough semantically for this. It is surely in the interests of software vendors to jump on the UML bandwagon and provide a bi-directional link between their interfaces and the UML interface.

Finally, I would suggest to all who teach system dynamics modeling to incorporate the concepts and practices of UML in their courseware and case studies. In this connection it would be useful to have a textbook which, using UML, describes the stages needed to reach the solution to a problem in the form of a system dynamics model. Such a book should include discussion of the stereotypes and generic

patterns specific to system dynamics. These activities can only help to encourage greater use of system dynamics.

---

[i] E.g. Sterman, J. D.: Business Dynamics, Systems Thinking and Modeling for a Complex World, Irwin McGraw-Hill, 2000, p. 42 ff

[ii] Booch, G., Rumbaugh, J., Jacobson, I.: Unified Modeling Language User Guide, Addison Wesley Longman, 1999

[iii] E.g. Rational Software Corporation: Rational Rose, Version 7.5; Quatrani, T.: Visual Modeling with Rational Rose 2000 and UML, Addison Wesley, 2000

[iv] Eriksson, H.-E., Penker, M.: Business Modeling with UML, Business Patterns at Work, John Wiley & Sons, Inc., 2000

[v] Ventana Systems, Inc.: Vensim DSS 32, Version 4.2a

Back to the Top