Go Back

Design Pattern Language and System Dynamics Components

(Preliminary Version)
By

Dr. Warren W. Tignor, Ph.D.

Vice President, Kimmich Software Systems, Inc.
7235 Dockside Lane
Columbia, Maryland 21045 USA
(410.531.1002 voice/ 410.381.5865 fax)
wtignor@ieee.com

1 Abstract

This paper explores the concept of design patterns and how they relate to System Dynamic models. Design patterns began as the province of architectural structure based on the work of Christopher Alexander. His thoughts found their way to software engineering as models for design patterns.

Like Alexander, Forrester is a strong advocate of structure. Forrester bases the foundation of System Dynamics models upon the cornerstone of structure. To him structure is essential if we are to effectively interrelate and interpret our observations in any field of knowledge. He has said that if one knows a structure or pattern on which he can depend, it helps him to interpret his observations.

Various System Dynamicists have described System Dynamics structural patterns in terms such as archetypes, templates, molecules, and components. However, the potential power of patterns as presented by Alexander has not swept the System Dynamics field based on these constructs. This paper takes a closer look at the elements of Alexander's design patterns and asserts that a pattern language for System Dynamics is the missing piece preventing the power of design patterns from helping System Dynamics have further reaching impact.

Introduction

A pattern to one person may be a primitive building block to another. For this paper, the point of view of a pattern is as a description of communicating structures that are customized to solve a general design problem in a particular context. To this end, the paper explores the structural concepts of "patterns" and how they relate to System Dynamic "models", "components", "molecules", and "archetypes".

Expert designers, regardless of field of expertise, know not to solve every problem from first principles, Vlissides, J., Helm, R., Johnson, R., & Gamma, E. (1995). It is beneficial to reuse solutions that have worked in the past. When designers find a good solution, they us it over and over; this is what makes them experts, Vlissides et al., (1995). A designer familiar with patterns can apply them to new problems without having to discover them.

Patterns have been long recognized in other disciplines as important in crafting complex systems, Vlissides et al., (1995). Christopher Alexander and his colleagues were probably the first to propose the idea of using a pattern language to architect buildings and cities, Vlissides et al., (1995).

Alexander, C. et al (1977) said, "Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice", (page *x*). Alexander et al. (1977) were talking about buildings and towns; but what he says is true of software design, Vlissides et al., (1995); and by deduction to System Dynamics models.

Forrester (1990) set the "cornerstone" structures of System Dynamics. Bruner (1960) suggests that understanding the structure of a subject is essential to understanding the subject. This paper will show that an understanding of the fundamental System Dynamics structures will allow Design Patterns to be related meaningfully using a pattern language.

2 Statement of the Problem

The potential power of patterns as presented by Alexander has not swept the System Dynamics field. This paper takes a closer look at the elements of Alexander's patterns and asserts that a pattern language for System Dynamics is the missing piece preventing the power of patterns from helping System Dynamics have further reaching impact.

3 Literature Review

The literature review is organized by the following categories: System Dynamics, Design Patterns, Design Patterns Language and Applied Systems. The first three categories are intended to provide a basis of comparison using foundation statements about each of the disciplines (System Dynamics, Design Patterns, and Design Patterns Language). The Applied Systems category represents a survey of literature using one or more of the foundation technologies (System Dynamics, Design Patterns, and Design Patterns Language).

3.1 System Dynamics

Forrester set the cornerstone for the structure of System Dynamics and it has stood the test of time. In Principles of Systems, Forrester (1990) states that structure is essential if we are to effectively interrelate and interpret our observations in any field of knowledge: "Without an organizing structure, knowledge is a mere collection of observations, practices, and conflicting incidents" (p. 1-2). It is the structure of a subject that guides us in organizing information. "If one knows a structure or pattern on which he can depend, it helps him to interpret his observations. An observation may at first seem meaningless, but knowing that it must fit into one of a limited number of categories helps in the identification. Structure exists in many layers or hierarchies. Within any structure there can be substructures", (Forrester, 1990, p. 4-1).

Likewise, Bruner (1960) tells us that it is the understanding of the structure of a subject that allows many other things to be related meaningfully. Bruner (1960) tells us that learning through the transfer of principles is dependent upon mastery of the structure of a subject. Understanding the fundamentals makes a subject comprehensible. Human memory is dependent upon structured

patterns for recall. Understanding the specific case of a structure is a model for understanding other things like it that one may encounter. The constant reexamination of material's structure results in a narrowing of the gap between advanced and elementary knowledge.

Forrester (1990) established the cornerstones of System Dynamic structure as the following: the closed boundary; feedback loops; levels and rates; and, within a rate, the goal, apparent condition, discrepancy, and action. Figure 1 presents these structure elements in hierarchical form (Forrester, 1990, p. 4-1):

- The closed system generating behavior within a boundary
  - The feedback loop
    - Levels as one fundamental variable type
    - Rates as the other fundamental variable type
      - The goal as one component of a rate
      - The apparent condition against which the goal is compared
      - The discrepancy between goal and apparent condition
      - The action resulting from the discrepancy.

Figure 1: System Dynamics Structure

The essential idea in a closed boundary system focuses on the interactions that produce growth, fluctuation, and change within the system. The boundary, Figure 2, "…encompasses the smallest number of components, within which the dynamic behavior under study is generated", (Forrester, 1990, p. 4-2).

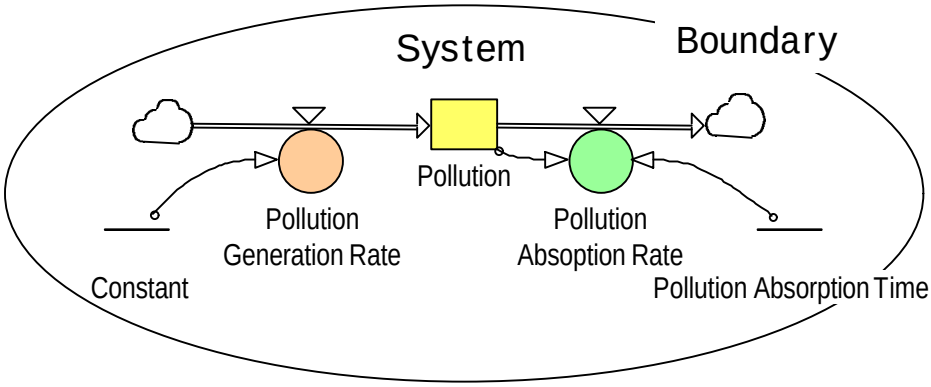


Figure 2: System Dynamics Boundary Concept

The feedback loop is the basic building block within the system boundary. The feedback loop couples the path connecting decision, action, system level, and information about the system level, with the final connection returning to the decision point, Figure 3.

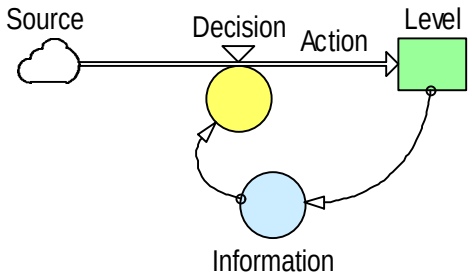


Figure 3: System Dynamics Feedback Loop

In this context, a decision process controls system action. The decision uses the available information to control action that influences the system level, and new information arises to modify the decision stream (Forrester, 1990).

Interconnecting feedback loops or a single feedback loop may constitute a system. At a lower level, feedback loops contain a substructure. Forrester (1990) tells us that there are two types of fundamental variable elements within each loop: levels and rates.

The level variable describes the condition of the system at any particular time. The level accumulates the results of action within the system. "The level variable accumulates the flows described by the rate variables", (Forrester, 1990, p.4-5).

In contrast to levels, the rate variable tells how fast the levels are changing. "The rate equations are the policy statements that describe action in a system, that is, the rate equations state the action output of a decision point in terms of the information inputs to that decision", (Forrester, 1990, p. 4-6).

Lastly, there is a substructure within a rate. According to Forrester (1990), there are four concepts within a rate equation: 1. Goal, 2. Observed condition of the system, 3. Discrepancy between goal and observed condition and 4. Statement of how action is to be based on the discrepancy, (p. 4-14), see Figure 4.

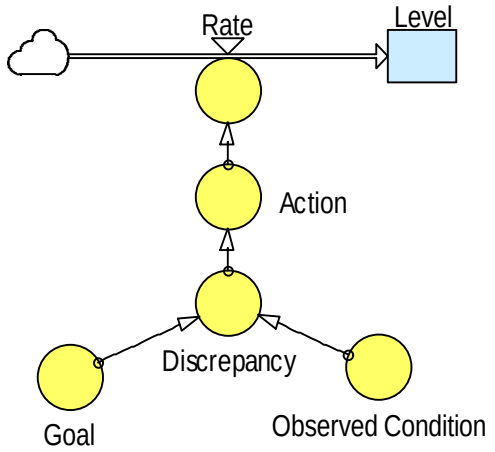


Figure 4: System Dynamics Rate Components

Forrester (1990) clarified that computing the successive time steps in the dynamic behavior of a system needed a standardized sequence for the computation and a terminology to use in

designating the procedure. He illustrated the computation in time steps as shown in Figure 5 below:

	← DT →			
	R20.JK			
L2.J	R2I.JK			
L1.J	R1.JK			
5	5+DT	5+2DT	5+3DT	6
J	K	L		
Time →				

Figure 5: System Dynamic Time Sequence

The time sequence figure assumes that the computations at time 5 were completed and the next computation period is 5+DT. The abbreviation DT stands for "difference in time", Forrester (1990). "The 5 and the 6 in the figure represent the units of time used in defining the system, for example, weeks or months, but the appropriate solution interval need not be the same as the unit of time measurement", (Forrester, 1990, p. 5-1).

The figure illustrated that there were four computations of system condition in each unit of time; "K" designates the current period of time and "J" the previous period of time. The levels L1.J and L2.J designate two values, system states, at the time "J". The rate R1.JK flowed into level L1. The rate R2I.JK flowed into level L2 while the rate R20.JK flowed out of level L2, Forrester, (1990).

3.2 Design Patterns

Design patterns describe the key ideas in the system, Fowler and Scott (1997). Patterns help explain why a design is the way it is. The design pattern represents the fundamental algorithm being implemented by the software, an algorithm that is repeated in many other designs. Vlissides et al., (1995) characterize design patterns as a description of communicating software that is customized to solve a general design problem in a particular context. The design pattern names, abstracts and identifies the key aspects of a common design structure that makes it useful for creating a reusable object-oriented design. Design patterns describe simple and elegant solutions to specific problems, Vlissides et al., (1995). Design patterns capture designs that have developed and evolved over time; they reflect extensive redesign and recoding as developers have striven for greater reuse and flexibility in their software, Vlissides et al., (1995).

Designing software is considered hard work; making it reusable is even harder. Vlissides et al., (1995) says that one has to find the pertinent objects, factor them into structures at the right granularity, define interfaces and inheritance hierarchies, and establish key relationships among them. The design needs to be specific to the problem at hand but also general enough to address future problems and requirements. Redesign is to be avoided if possible and minimized at the least.

Vlissides et al., (1995) say that a pattern has four essential elements:

1. The Pattern Name describes a design problem, its solutions, and consequences in a word or two. The name allows one to design at a higher level of abstraction. The vocabulary of

pattern names facilitates dialog. The name enables thinking about good designs and communicating them and their trade-offs to others.

2. The Problem describes the criteria for when to apply the pattern. Occasionally, the problem will include a list of conditions that must be met before it makes sense to apply the pattern.
3. The Solution contains the elements that make up the design, their relationships, responsibilities, and collaborations. The solution is not a particular concrete design or implementation but a template that can be applied to many different situations.
4. The Consequences are the results and trade-offs of applying the pattern. These are important for evaluating design alternatives and understanding the costs and benefits of applying the pattern.

The four essential elements above are part of the description of a design pattern that includes a graphical representation and also a record of the decisions, alternatives, and trade-offs that led to it; as well as concrete examples. Vlissides et al., (1995) advocate describing a design pattern using a consistent format based on the following Alexandrian template:

1. Pattern Name and Classification – the name is a metaphor for the design and the classification places the pattern in a taxonomy of patterns.
2. Intent – a statement of what the pattern does, rationale, issues addressed.
3. Also Known As – aliases for the pattern.
4. Motivation – a scenario of the problem and how the design pattern solves the problem.
5. Applicability – situations where the design pattern is applicable.
6. Structure – a graphical representation of the design pattern.
7. Participants – the objects participating in the design pattern and their responsibilities.
8. Collaborations – how participants carry out their responsibilities.
9. Consequences – addresses how the pattern supports its objectives.
10. Implementation – pitfalls, hints, or techniques that one should be made aware of before implementing the pattern.
11. Sample Code – software fragments that illustrate how one might implement the pattern.
12. Known Uses – examples of the pattern found in real systems.
13. Related Patterns – closely related design patterns, important differences, other patterns that work well with the one under consideration.

3.3 Design Patterns Language

According to Alexander (1979), a pattern language gives each person who uses it, the power to create an infinite variety of new and unique “buildings”, just as his ordinary language gives him the power to create an infinite variety of sentences. To him, each pattern is a rule describing what to do to generate the entity it defines; in this sense, the system of patterns forms a language. Alexander (1979. p. 183) says, “It is in this sense that the system of patterns froms a language.” “A pattern language is a system which allows its users to create an infinite variety of those three

dimensional combinations of patterns we call buildings, gardens, towns", Alexander (1979, p. 186). In summary: both ordinary languages and pattern languages are finite combinatory systems which "…allows the creation of an infinite variety of unique combinations, appropriate to different circumstances, at will", Alexander (1979, p. 187):

Natural language	Pattern Language
Words	Patterns
Rules of grammar and meaning which give connections	Patterns which specify connections between patterns
Sentences	Buildings and places

For Example, Alexander presents the pattern language for a farm house in the Bernese Oberland (1979, p. 187):

NORTH SOUTH AXIS
WEST FACING ENTRANCE DOWN THE SLOPE
TWO FLOORS
HAY LOFT AT THE BACK
BEDROOMS IN FRONT
GARDEN TO THE SOUTH
PITCHED ROOF
HALF-HIPPED END
BALCONY TOWARD THE GARDEN
CARVED ORNAMENTS.

Alexander says that each of these patterns, expressed in the form of a rule, is a field of relationships which may take an infinite variety of specific forms. In this sense, Alexander (1979, p 191) has found an example of the kind of "code" that, at certain times, plays the role in buildings and in towns similar to that the genetic code plays in a living organism.

He alludes that these kinds of languages are ultimately responsible for every single act of building in the world. To him, all acts of building are governed by a pattern language of some sort, and the patterns in the world are there, entirely because they are created by the languages which people use. The patterns come from the work of thousands of different people who build by following some rules of thumb, "…and all these rules of thumb–or patterns-are part of larger systems which are languages", Alexander (1979, p. 202).

Alexander asserts that every person has a pattern language in his mind; this seems similar to the Mental Models of System Dynamics! An individual's pattern language is the sum total of *their* knowledge of how to build. The pattern language in one person's mind is slightly different from

the language in the next person's mind; no two are exactly alike; yet many patterns, and fragments of pattern languages, are also shared, Alexander (1979).

Alexander (1979) believes that when a person is faced with an act of design, what he does is governed entirely by the pattern language which he has in his mind at that moment. The pattern languages are evolving all the time, as each person's experience grows. But at the particular moment one has to make a design, one relies entirely on the pattern language accumulated up until that moment. The act of design, whether humble, or gigantically complex, is governed entirely by the patterns one has in his mind at that moment, and one's ability to combine these patterns to form a new design (Alexander, 1979).

In general, every pattern must be formulated in the form of a rule which establishes a relationship between a context, a system of forces which arise in that context, and a configuration which allows these forces to resolve themselves in that context (Alexander, 1979, p. 253):

Generic form

Context → system of forces → configuration

Communal rooms →	conflicts between privacy → and community	alcove opening off communal room

Patterns provide an infinite number of solutions to any given problem. There is no way to capture the details of all these solutions in a single statement. It is up to the creative imagination of the designer to find a new solution to a problem which fits a particular situation. But when properly expressed, "…a pattern defines an invariant field which captures all the possible solutions to the problem given in the stated range of contexts", (Alexander, 1979, p. 261).

A pattern language has structure created by the fact that individual patterns are not isolated, Alexander (1979). In architecture, the patterns cover every range of scale. The largest patterns cover aspects of regional structure, middle range patterns cover the shape and activity of buildings, and the smallest patterns deal with the actual physical materials and structures out of which the buildings must be made. It is possible to put these patterns together to form coherent languages. Subsequently, each pattern then depends both on the smaller pattern it contains, and on the larger pattern within which it is contained. It is the "buildup" of patterns that result in the final structure.

"Each pattern sits at the center of a network of connections which connect it to certain other patterns that help it complete it", says Alexander, (1979, p. 313). The network of these connections between patterns creates the language. In the network, the links between the patterns are almost as much a part of the language as the patterns themselves, Alexander (1979). The structure of the network makes sense of the individual patterns; it anchors them and helps make them complete.

Each pattern is modified by its position in the language as a whole, according to the links which form the language. By virtue of its position in the whole, each pattern becomes intense, vivid, and easy to visualize. The language connects the patterns to each other, and helps them to come to life by giving each one a realistic context, and encouraging imagination to give life to the combinations which the connected patterns generate, Alexander (1979).

The language is a good one, capable of making something whole, when it is morphologically and functionally complete. A language is morphologically complete when the patterns together form a complete structure, filled out in all its details, with no gaps. It means that the general "species" of buildings which this language specifies can be visualized as a solid entity, whose only lack of clarity lies in the particulars.

It is functionally complete when the system of patterns has self-consistency. It is functionally complete only when all the internal systems of forces are completely covered-in short, when there are enough patterns to bring all the forces into equilibrium.

In both cases, the language is comepete only when every individual pattern in the language is complete. Every pattern must have enough patterns "under" it, to fill it out completely, morphologically. And every pattern, functionally, must also have enough patterns under it, to solve the problem which it generates.

3.4 Applied Systems

Corbin (1994) described a development technique for model conceptualization integrating archetypes and their corresponding generic models into a framework. He stated that model conceptualization was the most difficult stage of the modeling process and the most difficult to master. To conceptualize a model, the following was needed, see Figure 6 (Corbin, 1994):

1. The basic feedback structure
2. The level of aggregation
3. The model boundaries, and
4. The timeframe.

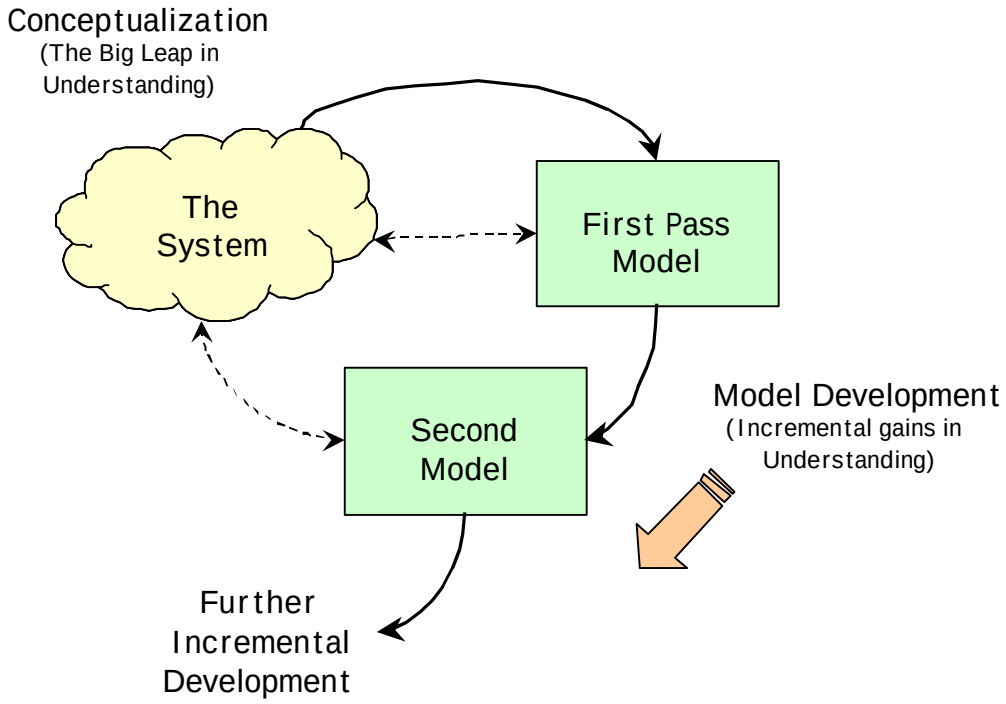


Figure 6: Conceptualization Modeling Process

Corbin (1994) identified three generic structures, see Figure 7, for further classification of their content:

Generic Models	Structures generic to a specific problem domain.
Archetypes	Structures transferable between different problem domains.
Building Blocks	Sub-structures found as building blocks in many different models.

Figure 7: Three Generic Model Structures

Corbin's (1994) conceptualization model used the "base" archetypes and generic models of those archetypes to transition to a simple working model. The "base" archetypes, see Figure 8, were based on the work of Wolstenholme and Corbin (1993):

		Intended Action	
		Control	**Growth**
System Reaction	**Opposition**	**B/R** Fixes that Fail	**R/B** Limits to Success
	Competition	**B/B** Fighting for Control	**R/R** Success to the Successful

Figure 8: Base Archetypes

Using the base archetypes, Corbin (1994) suggested basic steps for the conceptualization process as follow:

1. Specify the intended Behavior – Is the aim growth or control the intended behavior loop.

2. Identify the System Reaction – Will the system respond with growth or control.
3. Create the Base Archetype – Link the loops identified in 1 & 2 to create a base archetype.
4. Specify the Problem as a Generic Model – Take the simulation language specific model corresponding to the base archetype and customize it to represent the domain problem
5. Qualitative First Pass Model – Flesh out the loops with intermediate variables and organizational boundaries
6. Quantitative First Pass Model – Add extra detail to the model to keep it consistent with the qualitative model
7. Iterative model development – Develop the model from here on based on iterative simulation results.

One caution offered by Corbin (1994) when using this methodology based on archetypes was to strike a balance between the initial structure from the archetype and the danger of using an overly prescriptive structure that constricted the developers thinking about the problem, i.e., forcing the problem to fit the solution. In fact, Corbin (1994) felt that building the proposed framework around the full set of system archetypes would be too restrictive with the archetype not only being the starting point but also the ending point!

La Roche (1994) identified the concept of a template for the structure of a system dynamics model realized in the MicroWorld® software of DYNAMO PD+®. The Template Simulator was organized into four segments (La Roche, 1994):

1. MicroWorld
2. Infosystem
3. Controls
4. Coupling of process-chain and accounting.

The "Template-loops", La Roche (1994), comprised a very simplified structure of a business with subsystems that defined its behavior and profit:

1. A supplier with his own planning
2. A production process-chain
3. A production and supply control system
4. A sales operation trying to match backlog and market-driven delivery delay allowed.

La Roche (1994) discussed the concept of using scenarios for business-process-engineering as general tasks to maximize asset-turnover and net product contribution, depending on the type of demand variation the business was subject to. He identified fundamental types of process-chain control with application to the basic model (La Roche, 1994) as illustrated in Figure 9:

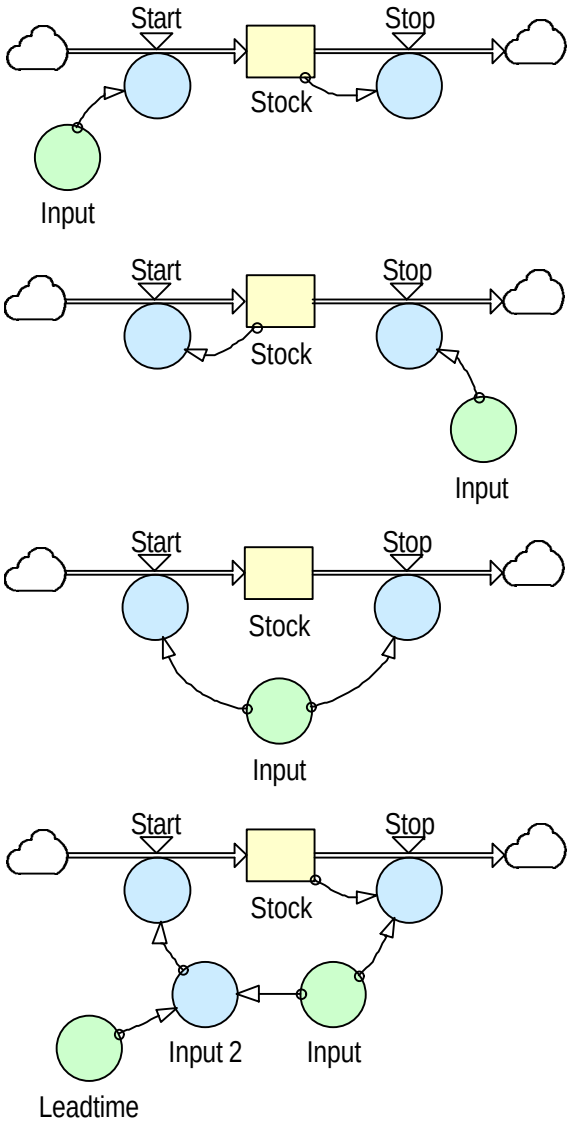


Figure 9: Process-chain control structures

La Roche (1994) said that using a template model at the start of the modeling process built on the essence of broad experience in the field. Templates lent themselves to an interactive and repetitive model building process (La Roche, 1994). Model building started with a provisional problem exposure of the people concerned with business process engineering using the template model and adjusting its parameters to fit the case at hand:

1. Expanding the template model structure towards the actual business process-chains.
2. Putting the pre-tested subsystems together as an updated version of the customized business-model.

La Roche (1994) believed that a continuous top-down model of the business-process-chain would be a useful and versatile tool to get the grand picture of the really worthwhile improvement of the process.

Joines and Roberts (1994)

Myrtveit and Vavik (1994) investigated modeling as a way of learning, and learning from running simulations. They found that to meet new requirements for learning environments that concrete objects were needed in addition to the general and abstract objects of accumulator-flow diagrams. Myrtveit and Vavik (1994) found that the use of objects was an elegant way to break the "world" into smaller parts that were easier to handle. To them, classification of objects was significant. Objects with the same properties (attributes) and operations were grouped into a class, Myrtveit and Vavik (1994). An attribute or operation local to a class was hidden (encapsulated) inside the class.

Myrtveit and Vavik (1994) thought that only in rare circumstances would an accumulator-flow diagram represent a natural object mapping a system. To them accumulator-flow diagrams focused on object attributes, and relationships between attributes. This focus was natural since the main purpose of accumulator-flow diagrams was to describe the dynamic relationships between attributes of a system and deduce the resulting behavior over time. To Myrtveit and Vavik (1994), providing higher level objects may be a way to make modeling useful to non-modelers.

Senge (1994) discussed seeing patterns of structure recurring again and again; he referred to these structures as archetypes and acknowledged that they recur in many different areas of knowledge: biology, psychology, economics, political science, management and ecology. To Senge, archetypes provided hope that specialization and fractionalization of knowledge would be bridged. Archetypes, per Senge (1994), were made of system building blocks: reinforcing processes, balancing processes, and delays. The structure of a frequently recurring archetype is illustrated in Figure 10, Limits to Growth, (Senge, 1994, p. 97).

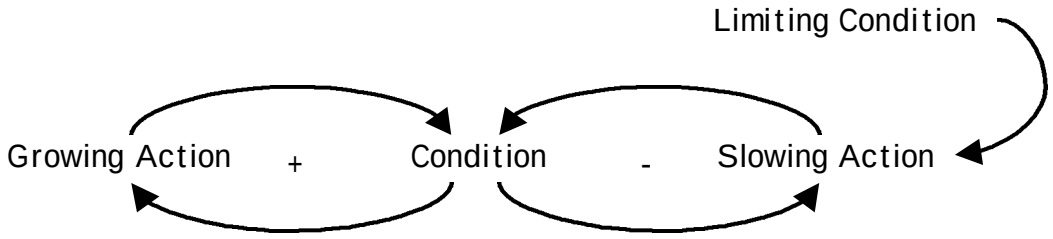


Figure 10: Limits to Growth Archetype

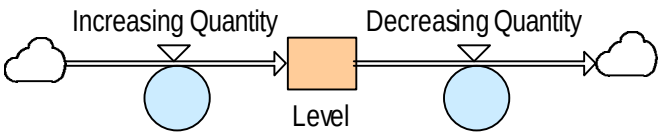
With the Limits to Growth archetype, a reinforcing (amplifying) process is set in motion to produce a desired result. A spiral of successes resulted; but they also created inadvertent secondary effects that eventually slowed down the success rate. According to Senge (1994, p. 95), the management principle of the Limits to Growth archetype is as follows: "Don't push growth; remove the factors limiting growth". Senge (1994) says that there is approximately a dozen system archetypes that affect us.

Eberlein and Hines (1996) published their first iteration of "molecules" that described fundamental System Dynamic capabilities. The molecules included the following:

1. Name
2. Parents
3. Used by
4. Category
5. Problem Solved
6. Equations
7. Description
8. Behavior
9. Classic Examples
10. Caveats
11. Technical Notes.

An example of a "molecule" is provided in Figure 11 below.

Name: Level (also known as "stock", "state", or "integration")



Parents: None
Used by: Present value, Cascaded level, Goal-gap or smooth, Level protected by level, Work accomplishment structure
Category: Fundamental structures
Problems solved: How to change incrementally, how to accumulate or de-accumulate, how to remember, how to remove simultaneity from a feedback look (see technical notes).

Figure 11: Molecule Illustration of a Level

Ahmed (1997) pointed out that traditional system dynamic software allowed users to build models from abstract primitives. He felt that this process was slow and required deep skills in both the discipline of modeling and the domain of the subject model. These two items, he claimed, were impediments to potential users and proposed a methodological process based on components that brought those with problem domain knowledge closer to the modeling domain without prerequisites of deep knowledge of the modeling process.

According at Ahmed (1997), component design was not new to software engineering; and with the increasing focus on object-oriented design, there was a great potential for reuse of parts or whole existing models. To this Ahmed (1997) focused on the specification of the requirements for component specification and design.

His work differentiated components from generic structures and molecules. Ahmed (1997) declared that a component was built from a collection of variables and a number of components could be configured into a model. To Ahmed (1997), components had two main features:

1. Specification, and
2. Implementation.

To Ahmed (1997), there were clear benefits to be derived from the use of components:

1. Enabling business professionals or engineers not trained as modelers to build models.
2. Shortening the development time and lowering the cost of producing models.
3. Allowing modelers to leverage each others components in their own works, and

4. Enabling model developers to concentrate on specific features of their models due to the availability of third party components.

To this end, Ahmed (1997) advocated a component catalog architecture, Figure 12.

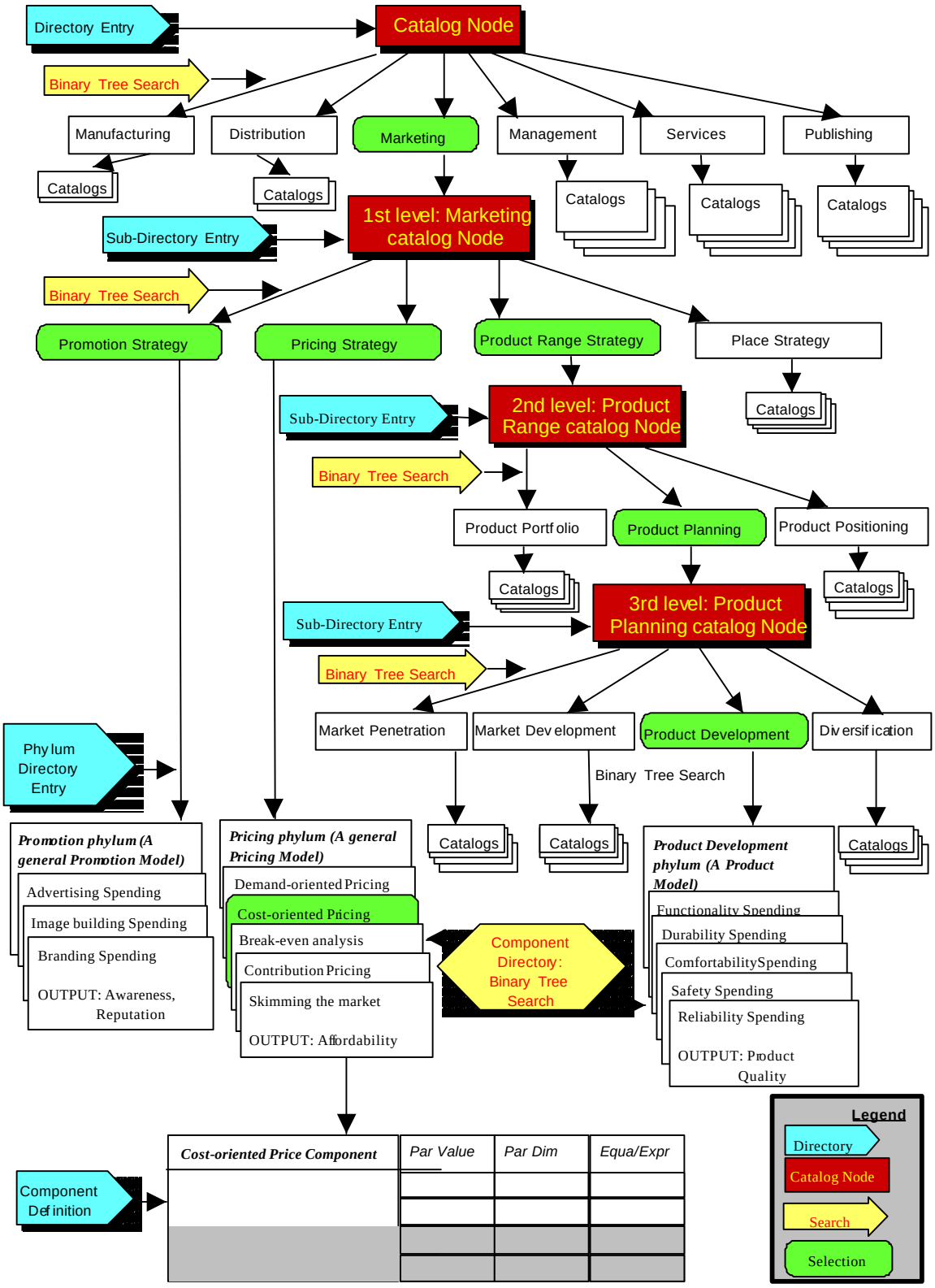


Figure 12: Component Catalog Architecture

Essentially, Ahmed (1997) said that modeling tools needed to be made available to practical business people, politicians and other professionals so that they could have an opportunity to learn to control unintuitive dynamic processes.

Kovács, Kopácsi, and Kmecs (1997) noted that software developers often experience the problem of creating components for an application that someone has produced previously. Without effective reuse tools, it is natural to create components from scratch than look for useful components in other programs or systems. In the field of flexible manufacturing systems and flexible manufacturing cells, this is often the case. Even though the components of flexible manufacturing systems and cells are the same types of machine tools, robots, and transfer equipment, the components are recreated rather than reused. Typically, they will differ from each other only in their amounts and working parameters, Kovács et al., (1997).

Myrtveit (2000) defines object-oriented extensions to the basic SD language of levels and flows. Basic SD has only built-in classes, called levels and auxiliaries. Myrtveit introduces user-defined classes, called components. The SD counterpart of an object is a variable. The submodel variable type is introduced to create hierarchical SD models, and to instantiate components. Links and flows are the SD counterparts of object relationships. Myrtveit extends this to wire connections, where parameters are bundled together into type-safe connections between variables.

4 Brief Description of the Research Method and Design

The research method compared the …

5 Data Analysis

Data analysis showed that "structure" is essential to simulation system design: Bruner (1960), Forrester (1990), Alexander et al., (1977), and Vlissides (1995).

6 Major Findings and Their Significance

The major findings of this research and their significance is presented as follows:

7 Conclusions

The major conclusions reached as a result of this research are as follow:

8 References

Ahmed, U. (1997). A process for designing and modeling with components. Proceedings of the 15th International System Dynamics Society. Turkey: System Dynamics Society.

Alexander, C. (1979). The timeless way of building. New York: Oxford University Press.

Alexander, C., Ishikawa, S., Silverstein, M., Jacobson, M., Fiksdahl-King, I., and Angel, S., (1977). A pattern language. New York: Oxford University Press.

Braude, E. (1998). Towards a standard class framework for discrete event simulation. Proceedings of 31st Annual Simulation Symposium.

Bruner, J. (1960). The process of education. Boston: Harvard University Press.

Corbin, D. (1994). Integrating archetypes and generic models into a framework for model conceptualism. Proceedings of the International Systems Dynamics Society. Sterling: System Dynamics Society.

Eberlein, R. & Hines J. (1996). Molecules for modelers. Proceedings of the International System Dynamics Society. Cambridge: System Dynamics Society.

Fayad, M. & Schmidt, D. (1997). Object-oriented application frameworks. Communication of the ACM40.

Forrester, J. (1990). Principles of systems. Portland: Productivity Press.

Fowler, M. & Scott, K. (1997). UML distilled: Applying the standard object modeling language. Reading: Addison-Wesley.

Joines, J. & Roberts, S. (1994). Design of object-oriented simulations in C++. Proceedings of the 1994 Winter Simulation Conference. IEEE.

Kovács, G., Kopácsi, S., & Kmecs, I. (1997). Simulation of FMS with the application of reuse and object-oriented technology. Proceedings of the 1997 IEEE International Conference on Robotics and Automation. Albuquerque: IEEE.

La Roche, U. (1994). A basic business loop as starting template for customized business-process-engineering models. Proceedings of the International Systems Dynamics Society. Sterling: System Dynamics Society.

Myrtveit, M., & Vavik, L. (1995). Object based dynamic modeling. Proceedings of the International Systems Dynamics Society. Tokyo: System Dynamics Society.

Myrtveit, M. (2000). Object-oriented Extensions to System Dynamics. Proceedings of the International Systems Dynamics Society. Bergen: System Dynamics Society.

Senge, P. (1994). The fifth discipline: the art and practice of the learning organization. New York: Doubleday.

Schöckle (1994). An object-oriented environment for modeling and simulation of large continuous systems. (Available at http://www.nmr.emblheidelberg.de/eduStep/…erences/OOCNS94/Proceedings/Schoeckle.html).

Taylor, D. (1990). Object-oriented technology: a manager's guide. Reading: Addison-Wesley.

Vlissides, J., Helm, R., Johnson, R., & Gamma, E. (1995). Design patterns: Elements of reusable object-oriented software. Reading: Addison-Wesley.

Wolstenholme, E. & Corbin, D. (1993). Toward a core set of archetypal structures. Proceedings of the International Systems Dynamics Society. Cancun: System Dynamics Society.