Hierarchical Approach in System Dynamics Modelling

Dong-Hwan Kim
Jae-Ho Jun
(Electronics and Telecommunications Research Institute, Seoul, Korea)

Electronics and Telecommunications Research Institute (ETRI),
6th-floor Leaders office Bldg 1599-11, Seocho-Dong, Seocho-Ku, Seoul, Korea.

TEL: 82-2-587-7472 (Office)
82-2-573-2192 (Home)
FAX: 82-2-588-7337 (Office)

Traditionally system dynamists used two kinds of modelling approach; top-down approach and bottom-up approach. With top-down modelling approach (feedback loop thinking) modellers first draw a causal loop diagram and then details it down to the stock and flow diagram and its equations. With bottom-up modelling approach modellers first draw stock and rate variables and then auxiliary variables.

Top-down approach forces a modeller to see forest rather than trees, while bottom-up approach forces a modeller to construct forest by planting trees incrementally. In general, systen dynamists use both approach in modelling processes simultaneously or sequentially. However, most of previous softwares for SD modelling have focused on the bottom-up approach. The 'operational thinking' concept of Barry Richmond is most typical in this trend.

In this paper, '*a hierarchical modelling*' is introduced as a new modelling approach which can integrate top-down approach and bottom-up approach. A hierarchical modelling approach extends the sector view concept proposed recently by Barry Richmond. In hierarchical modelling approach, a sector can contain many sub-sectors which also can have their own sub-sectors. A modeller can use a bottom-up approach or operational thinking in a sector. When a modeller wants to connect variables across sectors or to extend a variable into a sector, he can use a top-down approach.

In this paper, a hierarchical modelling approach is implemented with an object-oriented programming method. This paper explains why an object-oriented programming is necessary in implementing the hierarchical modelling approach and discusses some problems which should be resolved for developing a more easy and transparent hierarchical modelling approach.

Hierarchical Approach in System Dynamics Modelling

Dong-Hwan Kim
Jae-Ho Jun
(Electronics and Telecommunications Research Institute, Seoul, Korea)

In order to build a large model, I have replaced my computer screen with larger one. When I drawed a large SD diagram on a large screen, I realized that larger eyes and a bigger brain is also necessary. It was certain that something in my way of modelling was wrong, because I cannot update my small eyes and poor brain.

We think that traditional approaches for building SD models have two important flaws among other things. First is the missing link between the causal map and the stock-flow diagram. The second flaw is that traditional approaches are oriented to a single sheet world view, which says that every system structures can be represented on a single sheet. In this paper, we want to propose a hierarchical modelling approach as a solution to these flaws. A hierarchical modelling approach will not require a larger screen nor bigger eyes nor a better brain. This is implemented within the object-oriented programming environment.

First section reviews two kinds of traditional modelling approaches and their evolutions with the development of simulation softwares. Second section describes flaws and costs in traditional modelling approaches. Third section introduces a hierarchical modelling approach. In the fourth section, a hierarchical modelling approach is applied to the commodity cycle model of Meadow.

1. Evolution of Modelling Approaches

1) Top-down approach vs. bottom-up approach

Traditionally, system dynamists used two kinds of modelling approaches; a top-down and a bottom-up approach (Wolstenholme 1990; Coyle 1983). They are different at how one proceeds to construct a model. In the top-down modelling approach (feedback loop thinking), modellers first draw a causal loop diagram and then details it down to the stock-flow diagram and equations. In the bottom-up modelling approach (a modular approach in terms of Wolstenholme), modellers start with one or two key variables, particularly stock and rate variables.

Top-down approach forces a modeller to see a forest rather than trees, while bottom-up approach forces a modeller to construct a forest by planting trees incrementally. Recently system dynamists pay more attentions to the bottom-up approach. This trend can be summarized by the operational thinking concept of Barry Richmond.

> "Stocks and flows are very profound building blocks. They precede feedback loops. They form the infrastructure of a system. They provide the substrata for feedback loops to exist, just as the spinal cord and skeleton provide the framework for the muscles and organs that give rise to, and are stimulated by, feedback signals coursing throughout the nerves. Without the infrastructure, there can be no feedback system. ... Causal-loop diagrams, when used in advance of a stock-and-flow diagram, serve closed-loop laundry list thinking, not systems thinking. They represent the antithesis of operational thinking." (Richmond 1994, p.143)

We don't want to discuss on whether or not Richmond is right, but on how we can integrate the

top-down approach and the bottom-up approach. If the top-down approach (causal-loop diagram first) is the antithesis of the bottom-up approach (stock-flow diagram first), we must strive for reaching to a synthesis.

2) Softwares in favor of bottom-up approach

A decision on what approach to take may depends on individual preferences and educational backgrounds of modellers and on the kind of our knowledge about the system to be modeled. But it might be determined by the fashion which is shaped by the development of softwares.

It is certain that system dynamists use both approaches simultaneously or sequentially. However, most of softwares for SD modelling have focused on the bottom-up approach. A software for SD modelling should not be regarded as a simple tool. As computer scientists recognized, software is not soft. It is harder than hardware to its users. World views of the software users are shaped by the frame of the software. Illich points out how much important roles a tool play as follows.

> "Tools are intrinsic to social relationships. An individual relates himself in action to his society through the use of tools which he actively masters, or by which he is passively acted upon. To the degree that he masters his tools, he can invest the world with his meaning; to the degree that he is mastered by his tools, the shape of the tool determines his own self-image." (Fisher & Lemke 1988, p.2)

In fact, system dynamists have been affected by the shape of simulation softwares. Meadow warns that "our technical tools lead us to see the short-term, operational, quantitative, statistical, aggregate side of things, not the long-term, qualitative side, or the individual richness and diversity, or the most feasible visions of how things could be redesigned to work better" (Meadow 1985).

DYNAMO and STELLA which are most popular languages in SD society paid little attentions on the top-down modelling approach. They check causal loops only for avoiding a simultaneous calculation. In particular, with a powerful graphic user interface, STELLA set the less-proficient modeler to the bottom-up modelling approach. STELLA has contributed much to the popularization of system dynamics methodology. However, we should not avoid the negative side effects, because complaints on its flaws can generate new opportunities for double loop learnings of system dynamists.

Paradoxically enough, recent emphasis on the operational thinking triggered studies on causal loop analysis (Hall 1994; Eberlein & Peterson 1990). We think that these trends reflect balancing forces against the emphasis on the bottom-up approach.

2. Fundamental Flaws In Traditional Modelling Approaches

Recent emphasis on the bottom-up approach itself may not be a problem for a modeller. The critical problem is that we have no bridge between top-down approach and bottom-up approach. We have no tools even for linking a causal-loop diagram to a stock-flow diagram. The more fundamental problems inherent in traditional modelling approaches is that both of them implicitly constrain the conceptual world of system dynamists into two-dimensional plane.

1) Missing link between causal loop diagram and stock-flow diagram

Costs arising from the missing link between causal loop diagram and stock-flow diagram is two-fold. First is the cost for the top-down approach and the second is for the bottom-up approach.

First, those who take the top-down approach should build a SD model with papers filled with causal loop diagrams. Moreover, one should take a leap from causal loop diagram to SD model. To our knowledge, there are little guideline for this jump. Furthermore, paper working for drawing a causal loop diagram is often so tiresome that system dynamists draw it only on their mind. While building a model, the original causal map is lost out of sight.

Second, those who start model-building with a stock-flow diagram suffer lots of troubles which include the abyss between a mental model and the associated stock-flow representation, the visual complexity, and the ambiguity in the diagramming language (Richmond 1994).

Much more dangerous thing for the bottom-up approach is the modelling without understanding. Recent developments in software have made the process of building a model much easier (Machuca 1992). By simply walking through steps presented in the software manuals, one can build and simulate a model without understanding the system structure. This is the most dangerous side effect of developing user-friendly softwares.

2) Burdens for mapping hierarchical concepts into two-dimensional sheet

In a single sheet world view, one thinks that a system structure can be diagrammed on two-dimensional plane. However, it is reasonable to say that a system is structured within a three-dimensional space. Most psychologists accept that our mental models are composed as hierarchic structures.

If a model-building can be conceived as converting our mental models to SD models, a model-building in traditional approaches can be regarded as a task for translating hierarchical structures to horizontal chains of causal relationships. The phrase of 'abyss between a mental model and a stock-flow diagram' by Richmond describes this gap adequately (Richmond 1994). Visual complexity is a natural result of this gap between hierarchical mental model and horizontal SD diagram.

The cognitive map itself is originated from the analogy of the geographical map which translates three-dimensional spaces into two-dimensional paper. However, contrary to the geographical map, the cognitive map do not provides a tool for denoting the altitude of variables.

In the face of complexities provided by our environments and ourselves, we tend to simplify or abstract them into a conceptual hierarchy. H.A. Simon points out the hierarchic structure as an essential architecture of complexity.

> If you ask a person to draw a complex object - such as a human face - he will almost always proceeds in a hierarchic fashion. ... If there are important systems in the world that are complex without being hierarchic, they may to a considerable extent escape our observation and understanding. (Simon 1969, pp.217-219)

Social systems in our mental map is hierarchically organized, not only because they are empirically hierarchical but because we can perceive them only through hierarchical frameworks. There is an abyss between SD modelling frameworks and the hierarchical mental models which is deeper than Richmond described. In order to conceptualize and build a SD model, one should convert his hierarchical mental model into horizontal causal maps and stock-flow diagrams.

If one confines his attention to a single level in his conceptual hierarchy, he can get out of the burden for converting work. However, SD models tend to encompass different levels of a system such as an individual decision maker, business firms, and the market. In order to build a serious SD model, we need conceptual and technical tools for incorporating hierarchical structures into our SD model.

Nowadays, hierarchical modelling approach is proposed by various kinds of researchers including cognitive psychologists, AI researchers, theorists in system simulations, and computer programmers.

3. Hierarchical Modelling Approach as a Linking Pin

In this paper, '*a hierarchical modelling*' is introduced as a new modelling approach which can integrate the top-down approach and the bottom-up approach and can assist the hierarchical mental models. A hierarchical modelling approach extends the sector view concept proposed recently by Barry Richmond (1994). The sector view concept is one of the efforts for overcoming the limits of traditional modelling approach. However, it does not provide a complete solution to problems discussed above. Peterson anticipates another approach as follows.

> "The stock/flow framework for model conceptualization is rigorous and precise. It also is abstract, and in many cases is not a "natural" way for the less-proficient modeler to think about system structure. The layered approach outlined here offers one solution to this difficulty. Object-oriented programming concepts may form the basis for another approach" (Peterson 1994, 299-300).

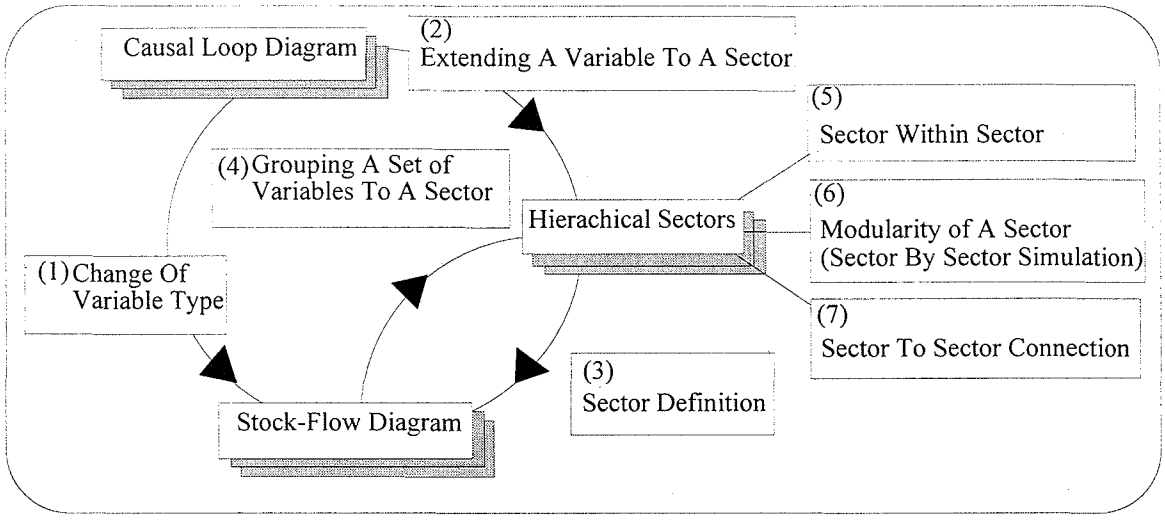
However, introducing object-oriented programming concepts in SD modelling will make a modelling task much more complex and difficult for the less-proficient modellers. If object-oriented programming concepts should be introduced into SD modelling, we think, it must be introduced indirectly.

The hierarchical modelling approach presented in this study is constructed within the object-oriented programming concept but has properties different from the latter. First, the hierarchical modelling approach does not require programming beyond the stock/flow framework. Second, as a result, it does not have a class hierarchy, an inheritance of methods, and encapsulations. The hierarchical modelling approach is only for modelling with hierarchical conceptualizations.

According to Simon, a hierarchic system can be defined as "a system that is composed of interrelated subsystems, each of the latter being in turn hierarchic in structure until we reach some lowest level of elementary subsystem" (Simon 1969, 196). In this study, I will adopt his definition for a hierarchical modelling approach.

In order to satisfy Simon's definition and to provide solutions to the problems discussed above, a hierarchical modelling approach should have functions summarized in Figure 1. In figure 1, seven functions are placed with three elements of a hierarchical model. Arrows in figure 1 indicates some of the plausible transformations among three elements.

For bridging the gap between the causal-loop diagram and the stock-flow diagram, a function for changing variable type (1) is introduced. Variables in the causal-loop diagram have no specific variable type. With the function for changing the type of variables, one can change a non-specified variable of the causal-loop diagram into a specific type such as level or rate.



<Figure 1> Functions necessary for a hierarchical modelling approach

However, it is a common case that a variable in the causal-loop diagram contains lots of the stock-flow diagram. In this case, one can extend a variable of the causal-loop diagram to a sector (2) in which a stock-flow diagram can be defined (3). When the stock-flow diagram becomes too complex to view, one can make a set of variables as one sector (4). These functions are available in any levels of sectors.

A key to interfacing between the causal-loop diagram and the stock-flow diagram is the potentiality of changing the type of variables, such as level, rate, and sector. In particular, the potentiality of changing a variable to a sector provides a linking pin between different levels of abstraction of the causal-loop diagram and the stock-flow diagram. This linking pin opens a way for hierarchical modelling methodology.

For implementing hierarchical model-building facility three kind of additional functions are required. First is a function for the requirement that a sector can have its own sub-sectors (5). This function satisfy the recursive definition of a hierarchical structure. The sector view concept of STELLA does not satisfy Simon's recursive definition of hierarchy, because a sector in STELLA cannot have their own sub-sectors. In hierarchical modelling approach, a sector should be able to contain many sub-sectors which also can have their own sub-sectors.

The object-oriented programming technology is necessary to implement 'the sector within a sector concept'. In the object-oriented programming environment, lots of sector object can be created as an instance of a sector class. Within the object-oriented programming environment, a programmer defines the sector class and some methods (functions) for allowing a sector objects to have its own sector objects. This is all that is necessary for implementing the recursive definition of hierarchical structure in the object-oriented programming. That is why I have made my SD software in the object-oriented programming environment; Smalltalk.

The second function for the hierarchical modelling is a modularity of a sector (6). If a content of a sector is dependent on the higher sector or on the entire model, a sector is nothing but a box for hiding visual complexities of the model. In the hierarchical modelling perspective, the content of a sector should be defined and modified without considering variables which are placed out of it. Only with the modularity of a sector, a hierarchical approach can simplify complexities inherent in modelling process. With the sector modularity, a modeller can develop

his model step by step as it becomes complex.

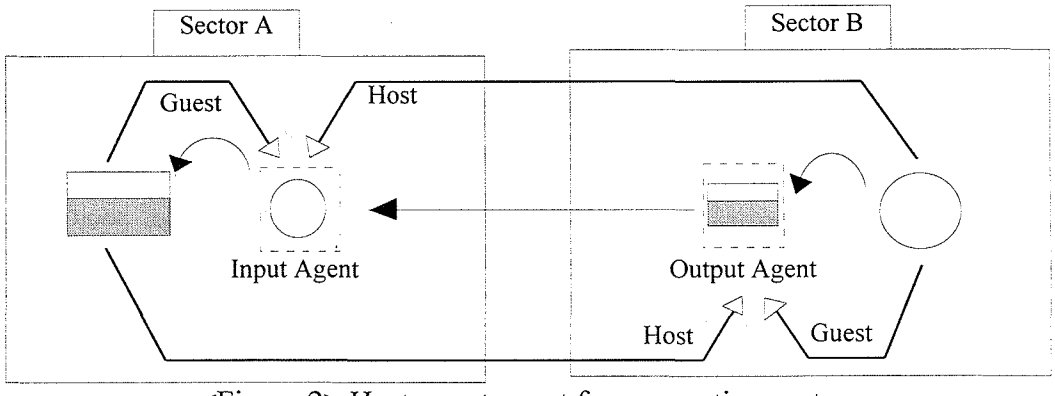
The sector modularity requires various sub-functions; importing a SD model into a sector, saving the content of a sector as an independent SD model, simulating and analyzing a sector without affecting other sectors out of it. Importing a SD model into a sector and saving its contents as an independent model can be easily implemented in the object-oriented programming environment as explained above. A simulation of a sector requires simulating all sub-sectors which are located in the sector, because behaviors of a sector depend on the contents of lower sectors. With these functions, one can simulate and analyze behaviors of a model with a bottom-up approach while building it with a top-down approach.

The third function required for hierarchical modelling is a function for connecting a sector to other sectors (7). This was one of the most difficult design problems in implementing hierarchical modelling approach. It must be designed to enhance visual representations of sector-to-sector connections and to satisfy the sector modularity.

To fulfill these requirements, an agent variable was introduced for connecting variables belonging to different sectors. The role of an agent has been explored within various kinds of researches including intelligent agents of human-computer interface.and a principal-agent theory of organizational economics. The agent plays for the host in specific areas where the host cannot play. Variables in a sector cannot play their roles out of their sector. They need their agent.

The agent is different with the ghost which was introduced in STELLA. A ghost is only a shadow of a real variable. If a real variable is deleted, the ghost must be deleted. However, an agent is an independent variable which can change its host.

Within a sector, a modeller should deal with agents which act for variables belonging to other sectors. He should know not only which agent represents which host but also which agent transacts with which variable belonging to other sectors. In order to represent their inter-relationships, we have introduced a basic mechanism of 'host-agent-guest' for connecting sectors. Their relationships are displayed in figure 2.



<Figure 2> Host-agent-guest for connecting sectors

Figure 2 shows a basic structure of connecting two sectors horizontally. When a modeller connects two sectors, an input agent and an output agent is created and inserted into the sectors according to the direction of a connection. In figure 2, a flow is connected from sector B to sector A. Therefore, an output agent is created in sector B and an input agent is created in sector A. The output agent is different with the input agent in that the former only receives a flow within a sector while the latter only originate a flow. Within a sector one cannot create a flow starting

from the output agent or ending to the input agent.

An agent variable plays the role of its host variable and interact with its guest variable. Note that the host of one becomes the guest of the other and the guest of one becomes the host of the other. In figure 2, host variable of an input agent is an auxiliary variable in sector B which is the guest variable of an output agent for sector B.

If a modeller connects other variable to the output agent, the output agent changes its guest to the variable, and it sends a message about the change to the corresponding input agent. On receiving the message from the output agent, the input agent change its host to the guest variable of the output agent.

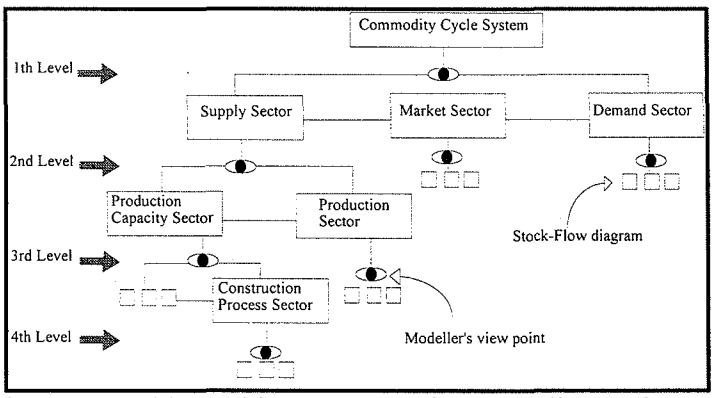
With this conceptual framework of host-agent-guest for connecting sectors, visual information about inputs and outputs of a sector are available to the modeller. A modeller can build a sector model without going out of it. Furthermore, because changes of hosts and guests for agents are automatically reflected on other sectors through interactions of agents, a modeller needs not pay his attentions to other sectors. This is what the modularity of a sector means.

Figure 2 shows how a sector-to-sector connection is implemented. As a matter of facts, one can connect a stock-flow variable to a sector and vice versa. In this case, only one input agent or output agent is created in the sector. It is because the host of the agent is fixed to the stock-flow variable.

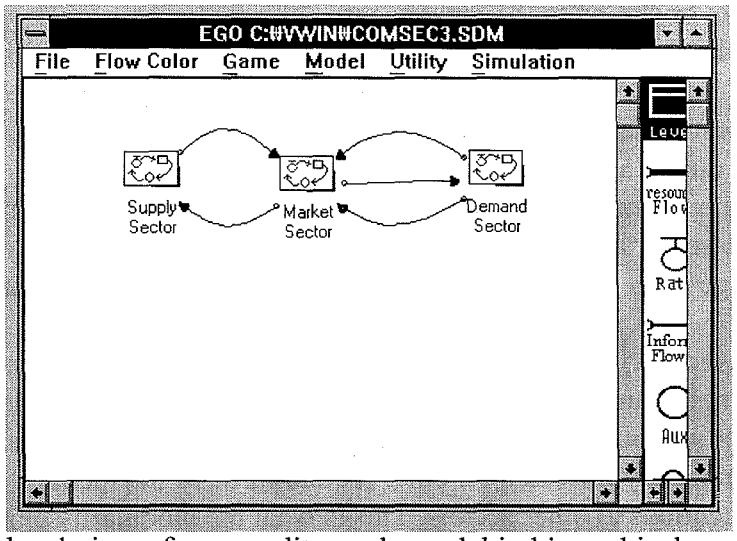
For the cause of hierarchical abstraction, one cannot delete agents in a sector. While agents belong to a sector, they are not defined within the sector. Connection of sectors which gives a life to the agent is a task belonging to the system above the sector. Only variables and equations defined in a sector can be changed or deleted within the sector. With the hierarchy of maintaining a model, one can get the modularity of a sector and the hierarchical simplifications and understandings of a complex system.

4. Hierarchical Modelling with Commodity Cycle Model

Conceptual frameworks and technological tools described above were implemented into EGO (Equations as Graphic Equations) software which was developed for building and simulating SD models (D.H.Kim 1995). In this paper, I want to demonstrate how one can apply the hierarchical modelling approach to the commodity cycle model of Dennis Medow. The hierarchic structure of the commodity cycle model is demonstrated in figure 3.

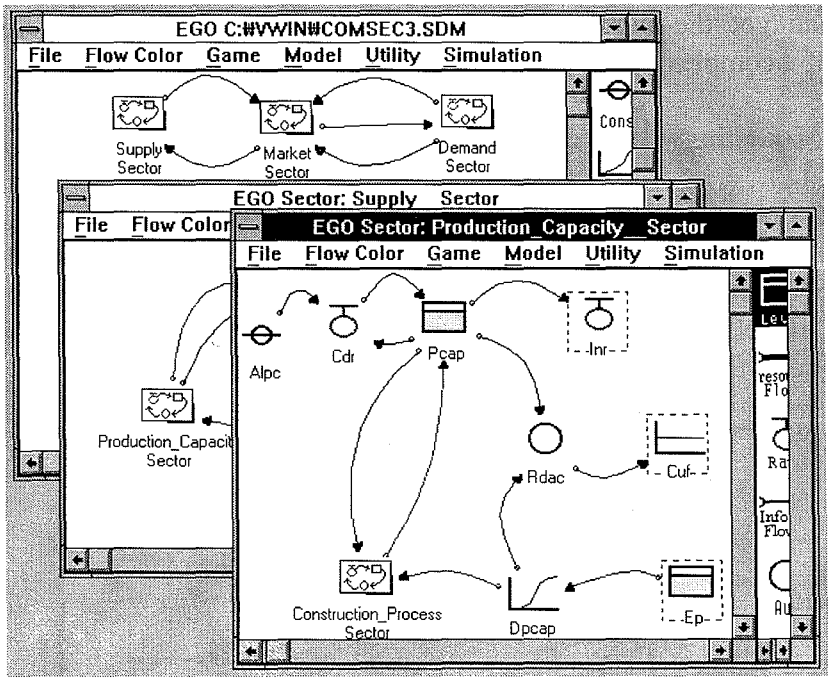


< Figure 3> A hierarchic structure of commodity cycle model



<Figure 4> Top level view of commodity cycle model in hierarchical modelling approach

With the top-down modelling approach, one can start to build three sectors in the first level of the hierarchy. It looks like figure 4 in EGO. Figure 4 can be regarded as an causal loop diagram which represent causal loops among supply sector, market sector and demand sector. One can open a sector by double clicking on it and can build a sector model. Figure 5 shows the screen view of Production_capacity__Sector which is placed in the third level of the hierarchy.



<Figure 5> Third level view in hierarchical modelling approach

As one can see in figure 5, the modelling tools and icons of a window for sectors are same as those of the first level windows. This shows that any tools and icons can be used for building and analyzing a sector. Even one can insert a model into a sector and save it into a separate file.

From this exercise, it is certain that the commodity cycle model is focused on the supply sector. One can elaborate the model by extending the market sector and demand sector. And this can be done by inserting into sub-sectors other models which are developed elsewhere to investigate dynamics of market or demand for a commodity.
As a natural consequence, one can easily construct a complex model with some of generic models. Generic models can be employed as an intermediate building block within hierarchical modelling approach. Within the traditional modelling approach, it is a difficult task to combine some of generic models. With the hierarchical modelling approach, we think, we should develop more simple generic models which can be fitted into sub-sectors.

5. Conclusions

There are some issues requiring further ideas and efforts. One of the most important issues is how can we improve the modularity of a sector. With the complete modularity of a model and their hierarchical structure, we can move into a new era of modelling complex systems as computer programmers did with the introduction of object-oriented programming technology.

In this paper, we presented a hierarchical modelling approach to integrate traditional approaches. We hope that it can make a SD model-building task become consistent to the way of building a mental model in our brain. We expect that the incorporation of hierarchical concepts into system thinking can give us a powerful tool. "If your only tool is a hammer, you treat everything as if it were a nail." If we accept this maxim, we should not discard our tool, but we must develop our tool fully enough that we can safely treat everything with our tool.

[References]

Coyle, R.G. 1983. The technical elements of the system dynamics. *European Journal of Operational Research.* 14:359-370.

Eberlein, R.L., D.W. Peterson & W.T. Wood. 1990. Causal Tracing: One solution to the Modeling Dilemma. *Proceedings of 1990 S.D. conference.* 341-354.

Fishcher G. & A.C.Lemke. 1988, Constrained Design Processes. in Raymonde Guindon (ed.) *Cognitive Science and Its Applications for Human-Computer Interaction, Lawrence Erlbaum Associates,* Publishers. pp.1-58.

Hall, R. I. 1994, Causal policy maps of managers: formal methods for elicitation and analysis. *System Dynamics Review,* 10(4).

Kim D.H. 1995, *Introduction to EGO.* Unpublished personal memo.

Machuca J.A.D. 1992. Are we losing one of the best features of system dynamics?. *System Dynamics Review.* 8(2).

Meadow, Donella H. 1985. System analysis and the world food system. *European Journal of Operational Research.* 20:158-167.

Meadows D.L. 1970, *Dynamics of Commodity Production Cycles.* Wright-Allen Press, Inc.

Peterson, S. 1994. Software for Model Building and Simulation: An Illustration of Design Philosophy. in J.D.W. Morecroft & J.D. Sterman (eds.) *Modelling for Learning Organizations.* Productivity Press.

Richmond, B. Systems thinking/system dynamics: let's just get on with it. *System Dynamics Review.* 10(2-3).

Simon, H.A. 1969. *The sciences of the artificial.* The MIT Press.

Wolstenholme, E.F. 1990. *System Enquiry: A System Dynamics Approach.* John Wiley & Sons.