Object-Oriented System Analysis of Anticipatory Systems in Weak Sense

Eugene Kindler
Ostrava University Faculty of Sciences
CZ – 701 03 Ostrava, Dvorakova 7, Czech Republic
FAX: +420 596 120 478
E-mail: evkind@barbora.mff.cuni.cz
http://www.osu.cz

Ivan Křivý
Ostrava University Faculty of Sciences
CZ – 701 03 Ostrava, Dvorakova 7, Czech Republic
FAX: +420 596 120 478
E-mail: Ivan.Krivy@osu.cz
http://www.osu.cz

Alain Tanguy
LIMOS CNRS UMR 6158, University Blaise Pascal, Clermont-Ferrand
Complexe scientifique des Cézeaux, F – 63177 Aubière, France
FAX: +33 473 405 001
E-mail: tanguy@isima.fr
http://www.isima.fr/limos/

Abstract
The authors implemented simulation models of anticipatory systems and a translator that converts models of conventional (non-anticipatory) systems into those of anticipatory systems in the weak sense. The systematic algorithmization leads the authors to recognizing some formal properties that could be viewed in a rather natural way for the anticipatory systems in the weak sense and that could be automatically converted into simulation models implemented on computer. The formal properties, i.e. certain concepts of system analysis and their professional names are presented in this paper.

Keywords: anticipatory systems, simulation, object-oriented programming, nested models, anticipation of anticipatory systems

1 Introduction

The system modelers know a lot of sorts of system, often hierarchically ordered. For example dynamic systems are classified into three sorts, namely into continuous systems, discrete event systems and combined continuous discrete event systems (shortly combined systems). The continuous systems are classified to those described by ordi-

nary differential equations and to those described by partial differential equations. The continuous systems described by the ordinary differential equations can be classified into a lot of sorts, among which there are compartment systems (Rescigno, Segre, 1961, Sheppard, 1962), electronic circuits (composed of idealized resistors, conductors, diodes, coils etc), systems mapping the processes in analog computers (composed of function generators, multipliers, integrators etc.) etc. The compartment systems can be further classified into tracer systems (reflecting mixing matter), into systems occurring in famous Forrester's "system dynamics" and into systems with chemical reactions. The discrete event systems are classified according to the dynamic of their structure – there is a large class of systems with transactions, i.e. with temporary elements, a smaller class of systems with the fixed number of their elements for that it is nevertheless possible to change their mutual relations, and a class of systems with the fixed structures, which is a smaller one but important, because it contains e.g. computers at the level of logical circuits or register transfers.

Such a classification is useful because of a lot of reasons, of which we can emphasize four ones: leading in analysis of a particular system, common terminology used by persons in their communication about systems, computer modeling of a particular system and software for computer modeling applicable for the systems belonging to a certain sort of the classification. During the last 50 years, the computers contributed to that task, because they need to get a classification and terminology that can be accepted by genial idiots (which is one of the metaphors used for computers), i.e. by automata that admit no intuition, experience and good will, which can arise among persons.

The state of the development of the research of anticipatory systems and their applications seems being matured to form a rather fine classification and a terminology related to the anticipatory systems. But – similarly as in case of other branches – neither the classification nor the terminology can be "decreed" by a person, a book or a paper, but can slowly develop in combining proposals and their wide accepting. Both the proposals and their accepting are supported by experience and applications.

The authors modeled various anticipatory systems in the weak sense (Kindler, Křivý, Tanguy, 2001a,b), (Křivý, Kindler, Tanguy, 2002) and studied formal properties of transforming a formal description of a conventional system to a formal description of its enriching that would be able to anticipate with an internal model (Kindler, Křivý, Tanguy, 2002a,b). The present paper contains definitions of terms, to which the authors came during their work when they communicate about the aspects of the anticipatory systems in the weak sense and of a more general theme of systems that carry models. The terms are proposed to serve for the initial steps in the classification and terminology.

Since several decades of years the authors have used object-oriented programming and namely SIMULA (Dahl, Myhrhaug, Nygaard, 1968), (SIMULA, 1989) and have continuously observed the great affinity between the object-oriented programming and the viewing on systems; the modern properties of SIMULA made it a suitable base of the classifying and terminology of the anticipatory systems in weak sense. Nowadays the object-oriented programming paradigm is accepted by the whole world community

of computer users and system professionals, and therefore its usefulness as the main stimulus for the classification and terminology (and for justifying it) is evident.

The principles of the object-oriented programming are not explained in details, because they can be supposed being so common and well-known in the human society of today like e.g. knowledge of writing. Only some terms will be introduced, in order to diminish the Babylon of the terminologies to which almost every school of the object-orientation contributes by presenting its own titles of concepts. Further synthesis of the object-oriented programming with other phenomena (agents, block, automatic translation) will be slightly mentioned as ways to the new knowledge and views covered by the proposed terminology and classification of anticipatory systems.

2 Tools for Description of Complex Systems

2.1 Object-Oriented Paradigm

The science of the present time is open to the exact studying of complex systems. These systems correspond to the real entities met in life sciences, technology and socials science. Computer modeling enables the description and the modeling of such complex systems by offering a lot of tools (databases, knowledge systems, simulation languages) to the human society.

Object-oriented programming appeared soon as a very efficient tool for expressing concepts. It was published by Dahl (1970), who was one of the inventors of that programming paradigm. The object-oriented programming enables to formulate concepts as *classes*, to order them according to the contents and extent and to introduce individuals that carry class content and are able to be modeled at computers. Such individuals are called *instances* of the corresponding classes. A class *A* can be classified as a *subclass* of class *B* if it is introduced by a statement that it has all properties and abilities introduced for class *B*. Then the instances of *A* are considered also as those of *B*.

The object-oriented paradigm corresponds to the organization of knowledge and concepts in every domain handled by a science, a technology or any organ for controlling the society. The first programming language oriented to that paradigm was SIMULA (Dahl, Myhrhaug and Nygaard, 1968), which has existed since 1968 almost unchanged until the present days. It was only slightly enlarged (Simula, 1989) but one can say that the computer program products elaborated e.g. in the 1969 can be used by the implementations prepared for personal computers and work-stations in the present millenium or at the end of the last one.

In the object-oriented paradigm, the contents of the concepts are reflected in the classes so that any class has its *attributes* and its *methods*. The attributes reflect the common quasi-static properties of the concept and those properties are "copied" as individual data structures of any instance of the class. For example, when the class *rectangle* has been introduced to have attributes *length* and *width* that reflect the dimensions of a general rectangle, then any instance of the class has the attributes called *length* and *width*; different instances can have different values of any of these attributes, but – moreover – these values of an instance (i.e. a rectangle) can change during the existence of the inst-

ance (the rectangle can "grow" or "diminish", etc.). But an instance cannot accept other attributes during its existence, i.e. attributes that are not introduced in class *rectangle* (that limitation causes that we use the words "quasi-static" above). Among the attributes, *reference attributes* can occur, the values of which point to some objects (instances of classes); these attributes reflect the structural relations among instances that are present in the same system.

Beside the attributes, the object-oriented paradigm supposes that the classes contain *methods*, i.e. algorithms, that reflect the dynamics with which the instances can influence a system, i.e. a community in which certain instances are present. An instance of a class A can be demanded to perform any of the methods introduced for A; in such a case, the instance performs the method and applies it to its own attributes. But the reference attributes enable an instance to work with attributes of other objects. Moreover, the methods can have parameters.

The described tools were designed in SIMULA at its very beginning phase and after more than ten years they stepwise penetrated into the programming practice in the world under the terms *object-oriented programming*, *object-orientation*, *object-oriented programming paradigm* etc., being built into the new popular programming languages (for example C++, SmallTalk) and into several languages that were not so popular. All those languages were called *object-oriented programming languages*. When one uses them to describe a dynamic system (i.e. a system the state of which changes in a time regarded as the Newtonian one, i.e. without relativistic effects) the dynamics should be decomposed into *message transfers*, i.e. into calling methods: a call sent to an object and asking it to perform a method μ is said a *message* to the object, the *selector* of which is μ). In other words, the dynamics of a system is initiated so that one of its elements performs a method that sends messages to other elements, when they perform the methods they send messages to other elements etc., and so a certain "avalanche" of methods rises, which reflects the complicated dynamics of the described system.

The tools of SIMULA applied in the paper (Dahl, 1970) referenced above belong to the object-oriented paradigm. Therefore, in principle, the other object-oriented programming languages are able to be tools for formulations of concepts. The concepts can form formal theories common in scientific branches studying theories, like general systems theories and mathematical logic.

2.2 Block Orientation and Local Classes

Much more than the object-oriented programming was implemented in SIMULA. It is namely the block orientation and the agent orientation.

Block orientation existed already in Algol 60 (Backus, 1960), then it was rejected by the theoreticians of structured programming in the 70-ies and since the end of the 90-ies it has very slowly returned, although it has existed without problems in SIMULA since 1967 and in its certain successor BETA since the 80-ies (Madsen, Møller-Pedersen, Nygaard, 1993). The block orientation enables a nesting of theories, i.e. formulation of concepts of entities that have their own theories, i.e. that have something like minds in which they can have their own concepts. In place of minds, we can consider (computer)

models. In other words, such languages enable to formulate formal theories with local concepts, i.e. with concepts that can be used only by some elements of the theories. The consequence of it is that one can describe (and then model) systems containing elements that carry models of systems. In other words, the consequence is that the object-oriented programming languages that are also block-oriented can serve for formulation of concepts concerning the systems in which models of other systems are nested. These models can reflect a system similar to that in which they are, or a system rather different. The first case may exist in case that one describes an anticipatory system of the weak sense, i.e. using its own model for getting information supporting the anticipation; that model is the nested (internal) one.

Theoretically, one can neglect the measurable flow of time. Thus the object-oriented languages that are also block-oriented can be used for the modeling of anticipatory systems of the weak sense. Nevertheless, in practice the measurable flow of time is not negligible. And in such a situation, the internal model should be a simulation one. Note that certain simulation aspects exist also when the internal model is not implemented at computing technique; for example the internal model that exists in the mind of a person who is inside a certain system and anticipates about it, often respects the rule demanding the same ordering of events in the modeled system and in its model – such a rule is essential for simulation models.

2.3 Quasi-Parallel Systems

For the formulation of concepts concerning the simulation of systems that have simulating elements one needs to use programming languages that admit "life rules" as components of classes. Such life rules have a form similar to that of conventional algorithms and are executed when an instance of a class arises (is generated, enters the studied system). An execution of such an algorithm is called ***process***. The tools common in conventional algorithms and applied for adapting the algorithm to the instantaneous situation – like jumps, branchings and cycles – can occur in the life rules and can cause that the processes corresponding to different instances of the same class behave differently, contrary to the fact that they "live" according to the same life rules.

The "lives" of the components of a real system can progress in a parallel way. The processes that model the lives at a conventional (i.e. monoprocessor) computer cannot run in a parallel way and therefore the computer switches among them so that the observer of the computing gets an illusion that the processes are performed in a parallel way. The system of those processes is called ***quasi-parallel system***.

For a real system, one supposes a ***real time*** in which it exists or should exist. In case of simulation the corresponding model should exist as a part of the existence of a certain real thing called ***carrier*** (of the model) that exists in the same real world as the real system and that computes in the same physical phenomenon which is the real time as that, in which the real system develops (but not necessarily ***during*** the same time interval). The carrier can be e.g. a person who imagines the future, or – more frequently – a computer. When the simulation model works and when the carrier is a computer, a quasi-parallel system exists at this carrier.

Contrary to the fact that certain sorts of quasi-parallel systems were introduced already in the first discrete event simulation language GPSS (Gordon 1961, 1969) and that the authors of SIMULA built the general tools for forming quasi-parallel systems into the object-oriented already in 1967 (Dahl and Nygaard, 1968), only a small number of every object-oriented programming language admit them. Beside SIMULA it is only BETA, JAVA and MODSIM. Unfortunately, MODSIM is not block-oriented. But also the other two languages suffer by certain obstacles. BETA demands a rather strong discrimination between the elements that belong to a quasi-parallel system, and the other ones (Vrba, 1999); when that discrimination is mapped to the description of the simulated system, the result is that an element that actively lives (i.e. according to its life rules) and is assigned with a certain name used in the simulated system, cannot have this name when it ends its active life (e.g. when it exhausts all its life rules). It is a serious obstacle for applying BETA.

A bad situation is with JAVA, too (Brassel, 2001). In (Vrba, 2000) examples are presented, illustrating that in this language the quasi-parallel systems do not behave deterministically and therefore that the computer simulation cannot be reproduced.

When one describes a (may be real) system in JAVA it is admitted to include description that concerns only the corresponding model, i.e. that concerns something existing only in the internal physical reality inside of the computer and that has no mapping in the modeled system. Against it, SIMULA is an ideal tool for exact description of the systems, because – beside others – its tools for representation of concepts do not allow to penetrate into their computer representation. In other words, when one describes a system that should be modeled SIMULA does not allow him to express anything that happens in the corresponding computer model.

When the concept of a process is opened for the manipulation offered by the object-oriented paradigm, it becomes ***agent***. When the life rules of such an agent enter a block in that classes are introduced, it becomes and ***intelligent agent***. Such a block represents a model owned by the agent or a phase in which the agent "thinks", using the concepts represented by the classes introduced in the block. Therefore it is possible to say that the optimal tool for exact analysis of the complex system and especially for the anticipatory ones in weak sense are languages with three orientations, i.e. object orientation, block orientation and process (or agent) orientation. We can say that SIMULA is as good tool for analysis of the anticipatory systems as e.g. theory of sets for pre-computer mathematics.

3 Terminology

3.1 General Case

For the next explication, it is necessary to introduce some concepts and their names.

There is a ***real system*** that should be modeled. It could be also called ***system of the first regard***. This system either exists in the real world or is supposed to exist there. In case of conventional modeling (including conventional simulation) it is modeled at a device (e.g. a computer, may be also a brain or a mind) called ***carrier of the first***

regard, which exists in the same real world but can exist not only contemporarily (it is a rare case) but also later and – most frequently – sooner than the modeled system. This carrier carries a model of the system of the first regard; this model is called ***model of the first regard***.

In the preceding section, we already introduced that the real system exists in ***real time*** and that it can have a ***carrier*** of a model of certain system S; let S be called ***system of the second regard***, the mentioned carrier be called ***carrier of the second regard*** and the model carried by it be called ***model of the second regard***.

The system of the second regard can have also a carrier of a model of a system; let they be called carrier ***of the third regard***, ***model of the third regard*** and ***system of the third regard***. So one can continue to systems, models and carriers of higher regards.

In case of simulation the following relation holds:

Let M be a model of a system S of the n-th regard. Then M is a model of the n-th regard and it exists at a carrier of the n-th regard in a real time. The states of the simulated system should have analogies at certain states of the model. Therefore some moments (of the real time), in which the carrier exists correspond in this manner to the moments (of the real time), in which the system exists. In other words, there is a certain set D of moments in that the model exists, so that for any element t of D a certain moment T of the simulated system exists so that the state $s(t)$ of the model at time T corresponds to the state $S(T)$ of the simulated system at time T. The state $s(t)$ can be enriched of the value of T. That value is called ***simulated time of the n-th regard***. Note that D does not need to cover the whole interval of real time moments during which the model exists (e.g. at a computer a lot of moments can exist during the time when it simulates, of which it is computing some auxiliary values for the simulation). The simulated time can be interpreted also for the elements of D: if t is an element of D and T is the simulated time for $s(t)$, then the simulated time at t is defined as $\tau(t)=T$. For the case of simulation, $t_1<t_2$ implies $\tau(t_1)\leq \tau(t_2)$: "simulated time cannot descend".

3.2 Special Case

The theory admits that the number of the regards is not limited. Also the practice admits it – often one anticipates in the way as "if I wish know something on future reactions of my partner X, I must take into account that X would know something on the future reactions of his partner Y so that Y would take into account what he expects about his partner Z ...". In other words, a human can anticipate using a model, in which an anticipation of another person is included etc. The models of such systems were already made (Blümel, Kindler, 1997).

Nevertheless – in a common practice – anticipation using computer simulation models is rather limited. The usual manner of the present years does not overpass the first regard. In that case, the determinations "of the first regard" are useless and one can speak on a ***real system***, its ***model*** and the ***carrier*** of it, and on the ***simulated time*** occurring in the model.

For the present development of the informatization of the human society, anticipation using simulation models of two regards becomes actual: a human-made system is

designed so that the anticipation of its behavior is simulated, but it is often evident that computers will occur in this system to help for temporary decisions: by means of simulation the computer will anticipate certain consequences of the decisions formulated during the existence of the system (i.e. a rather long time after having finished the design of the system). The first examples of such a method already exist (Blumel et al., 1997, chapter 4), (Kindler, 2000).

In such a situation there are exactly two regards and the words "of the first regard" can be replaced by the adjective ***external*** and the words "of the second regard" by ***internal***. Therefore the ***external system*** is simulated by an ***external model*** realized at an ***external carrier*** and the same external system contains an ***internal carrier*** among its own elements so that at the internal carrier an ***internal model*** is implemented that simulates an ***internal system***. The real time of the external system is mapped as the ***external simulated time*** in the external model and the real time of the internal system is mapped as the ***internal simulation time*** in the internal model.

Let our next considering be oriented to the case just introduced. Such case of simulation is called ***nested simulation*** or ***nesting simulation***. As the internal model is a part of the existence (of the "life") of its carrier both the internal simulated time and the external one are meaningful inside the internal model. As an example the following phrase can serve: during the time interval $<T_1,T_2>$ the internal model simulates what could happen in the internal system during time interval $<t_1,t_2>$; T_1 and T_2 speak on the external simulated time and t_1 and t_2 speak on the internal simulated time.

Theoretically an external system can have several (external) carriers, moreover, the carriers can dynamically arise (enter the external system) and disappear (leave the external system). In practice, we can expect that in the near future only one carrier will be in any external system. Let this case be called ***simple nested simulation*** and let us limit the following consideration to it. The carrier can carry several internal models. In time sharing mode, they can exist at the same carrier contemporarily. Otherwise they can alternate. It is evident that the usual situation is that there are phases when the carrier exists without carrying an internal model. The internal models existing at the same carrier can be rather similar (differing only by some parameters) or rather different. In the first case, we can speak about ***homogeneous simulation*** and in the second case about ***heterogeneous simulation***. But instead of using term homogeneous simulation, one uses term ***reflective simulation***. Examples of reflective simulation are in (Kindler, 2001) and (Kindler, 1994).

The border between the heterogeneous and reflective simulation is not clear; one of the reason is that the external model must differ from that internal – otherwise the internal model should reflect the situation that it simulates a system with a carrier of another model (that of the third regard), that model simulates a system with a carrier of another model (that of the fourth regard) etc. until infinity. That the clear cases of heterogeneous simulation will be those where the internal model simulates a ***fictitious system***, i.e. a system for which it is evident that it will never exist in a material way. The models of such fictitious systems can well replace some routines. Several examples are presented in (Kindler, 1995).

Beside the reflective simulation and heterogeneous one, it is suitable to introduce a rather interesting and strange sort that could be called ***simulation homogeneous-plus***. In it, the internal system reflects in good details the external one but contains other components that could belong to a fictitious system. An example (together with an application) is presented in (Novak, 2000).

4 Discrimination Between System Entities and World Views

Let us introduce ***entity*** as a common concept covering systems and their components. In every case of system analysis one needs to give names to the entities. That was reflected in all programming languages that enabled to handle with data structures; such data structures were viewed as primitive images of entities. Naturally, giving names was preserved for more sophisticated programming languages, namely for those that offered to represent entities as objects. The rather old languages had very poor tools for mapping entities. The famous programming language ALGOL 60 (Backus, 1960), admitted to give names only some parameters of procedures; its greatest contribution to the development of the computer programming – the blocks – could not get names, because they were not considered as entities but as something that could be characterized as ***world viewings***. In fact, there is a certain "metaphysical" difference between the systems and the views to them and therefore it is not possible to consider the world viewings exactly as entities.

One of the most excellent Simula property is that it takes the mentioned difference into account and that it makes two differences between the blocks and objects: while the objects can get names the block must be "anonymous"; and while the blocks can fully handle with quasi-parallel systems that handling is so limited for the objects that they cannot represent world views in which entities can contemporarily exist and fully interact in the common Newtonian time, i.e. along a time axe proper to the object. Therefore, in a non-degenerated case of modeling, the view to a world in which entities exist and interact in the Newtonian time cannot be considered as an entity occurring in another block, which represents also a view to a world governed by a Newtonian time flow. Nevertheless a block can contain an entity representing a carrier of a model, i.e. an entity, among the life rules of which there is a block that represents another world viewing than that representing by the block in that the carrier occurs. See Figure 1, where the

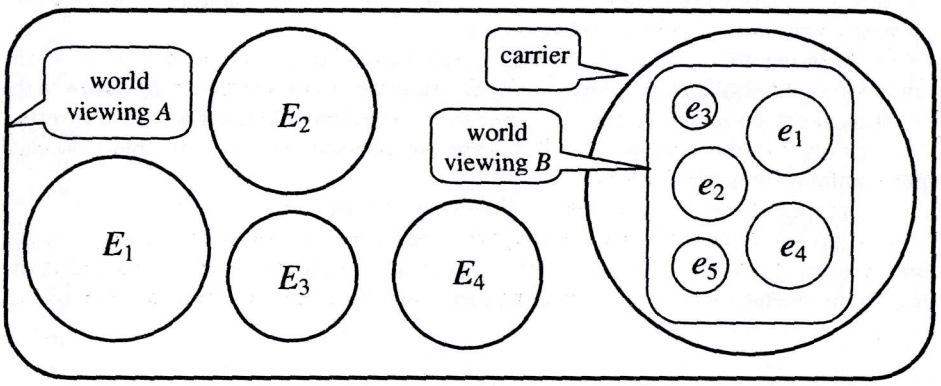

Figure 1: Worldviews

blocks are represented by rectangles with rounded edges while the entities are represented by circles; in the world viewing A there are four entities E_i (i=1,...,4) and a carrier of the world viewing B in which there are five entities represented by circles e_j (j=1,...,5). The consequence of that nesting of a block inside the object is that the world viewing corresponding to the block is an internal ("private") phase of the carrier corresponding to the object; the world viewing cannot be accessed and manipulated from the environment of the carrier but the same world viewing can "regard" to that environment and behave according it.

The discriminating between the worldviews and the entities could seem to be a restriction of the language that could cause problems of its application. But it is not true, and the good influence of the discriminating operates in the system analysis. Penetrating into a world viewing from outside is illogical and the discriminating – combined with the rules of the object-oriented programming – protects against it. In other words, the absence of the discriminating would allow expressing such obscure phenomena as e.g. a "telepathy" between two minds or between two computers (or between a mind and a computer) – although such phenomena would be interesting in the parapsychological sciences they are excluded from system analysis: there they cause inconsistencies (Kindler, 1998).

5 The Activities of Objects as Objects

Let us present a simple overview of the ideas derived in the preceding parts:

Consistently with the remarks made in 2.1, the conventional object-oriented paradigm does not reflect nesting models. As it reflects the human thinking on the systems it supports the opinion that the system analysis works with the following categories:

5.1. Abstract categories of ***classes*** that are ordered by means of ***specialization***, which is a relation among a pair of classes: one of them has greater contents that the other and is called its ***subclass***. The class contains ***attributes*** and ***methods***.

5.2. Categories directly present for the particular systems are ***objects***, which are ***instances*** of classes. Every instance of a class has its own dose of the attributes and is able to execute ***activities*** introduced as methods for the class.

The relation between methods and activities is similar as that between classes and objects; to a class more instances can exist contemporary, and – similarly – to a method more activities can be called contemporarily.

Overpassing the boundaries of the conventional object-oriented paradigm in the direction to the anticipatory systems in weak sense, one must add that of ***life rules*** to the mentioned categories, i.e. a category inspired by simulation languages that also reflect the thinking on the systems and that were predecessors of the first object-oriented programming language SIMULA.

The nesting models carry another category – ***world viewing***.

It could seem that the categories just mentioned are basic ones for the analysis of the anticipatory systems in the weak sense, because each of the categories is founded at the algorithmic behavior of the digital computers, and the ranging of every system of that

sort can be founded at the concepts introduced in chapter 3. Nevertheless there is a certain complication. Let us introduce it.

Consider a general process t of transformation of any conventional system S to an anticipatory one S^* in a weak sense. Suppose S is analyzed under a world viewing W. In reality such a transformation takes S, gives it one or more carriers and facilitates them by world views w analogous to W and by abilities α to construct internal models μ according to the instantaneous state of S. The carriers can form a class κ which forms a component of a world viewing W^* that is an enlargement of W.

The process t is a mental process and can be considered as a higher level of system analysis, because it does not concern an analysis on one particular system but that of an infinite set of systems. The mental process t can be modeled as a machine translation τ of a formal description of any conventional (i. e. not anticipatory) system S and world viewing W corresponding to it into a formal description of a corresponding anticipatory system S^* and to the world viewing W^* corresponding to it. We started to implement τ by means of SIMULA (Kindler, Křivý, Tanguy, 2002a,b), i. e. τ is written in SIMULA so that it reads texts in Simula and converts them to other texts in SIMULA.

A habitual view to the activities is that they are anonymous and therefore do not get names, contrary to the fact that the naming of methods is considered as a necessary aspect of the object-oriented programming. In other words, analysis of conventional systems does not accept the naming of activities, they are to be anonymous. That principle has been built into SIMULA. The consequence is that the local data of the activities are not accessible from the outside of the activity. The work at t discovered obstacles that the mentioned anonymity of the activities causes. They can be explained in the following example:

Suppose that the life rules of an object A force it to send a message to an object B, telling it to execute a method m. B executes m, i.e. causes an activity C to arise; but the method m is formulated so that it sends a message to an object D, demanding it to execute a method n. Therefore in a certain moment an "operating chain of the object A", i. e. the set of A, C and G, where G is the activity generated by the method n exists in the system model (see Figure 2). All the three members of the operating chain may have their own local data (schematically represented as $L(C)$ and $L(G)$ in Figure 2). Suppose all that to have come shortly before it is decided that an internal model should be con-

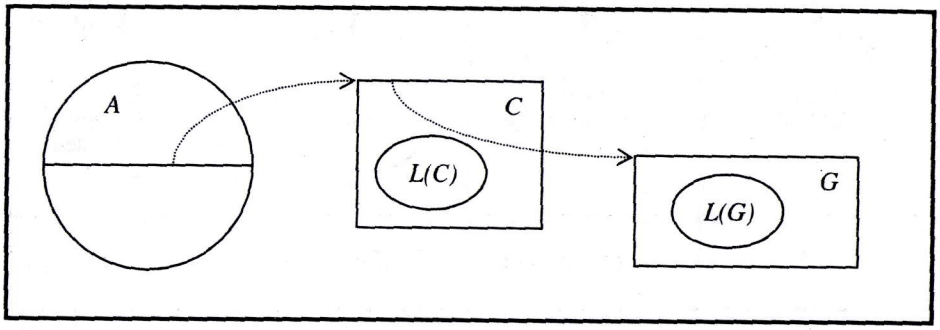

Figure 2: Operating chain of object A and two activities C and G

structed and applied. Then the initial state of the internal model, which should "copy" the instantaneous state of the external model, should also form a copy of the operation chain of A, in order that the image of A in the internal model could continue in the internal model exactly in the same manner as A would have continue.

The consequence of the anonymity of the activities is that the local data of C and G cannot be copied into the internal model, because the names C and G can be used in the present paper but cannot be assigned in SIMULA description.

But one does not need to blame SIMULA. On the contrary, that language leads us to a new decision on the system analysis. This decision is that in place of the methods related to any class H of W classes local to H are considered and therefore instead of the activities, corresponding instances of the local classes should be considered. Then the objects can get name and their copies (including their local data) can be formed in the internal model.

Therefore the analysis of the anticipatory systems in the weak sense can be made by the categories presented in 5.1 and 5.2, excepting the pair methods-activities that should be replaced by classes-instances (see Figure 3).

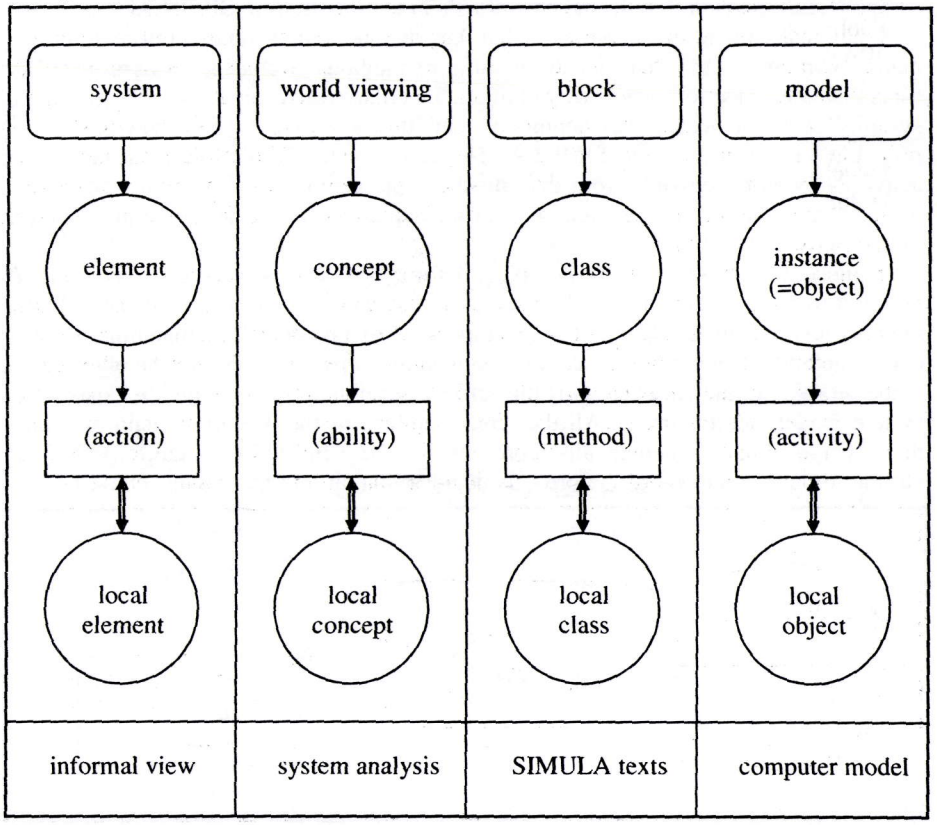

Figure 3: Simple arrow represents relation of locality, doubled one represents replacing

6 Conclusion

The analysis made by the technique described in this paper leads directly to simulation models, which can be generated directly from the description of the analyzed system in the programming language SIMULA, and – moreover – the exact formulation of the world viewings on the anticipatory system can be automatically generated from the world viewings oriented to the corresponding conventional (non-anticipatory) systems. Then the resulting description of the world viewing can be used for a simple description of anticipatory systems and for automatic implementation of their simulation models.

References

Backus John W. et al. (1960) Report on the Algorithmic Language ALGOL 60. Numerische Mathematik, 2, pp. 106-136.

Blumel Eberhard. et al. (1997) Managing and Controlling Growing Harbour Terminals. The Society for Computer Simulation International, San Diego, Erlangen, Ghent, Budapest.

Blümel Peter and Kindler Eugene (1997) Simulation of Antagonist Mutually Simulating Systems. Edited by Oliver Deussen and Peter Lorenz. Published by The Society for Computer Simulation International, Erlangen, Ghent, Budapest, San Diego, pp. 56-65.

Brassel Kai H. (2001) Advanced Object-Oriented Technologies in Modeling and Simulation: the VSEit Framework: ESM2001 – Modelling and Simulation 2001, Proc. 15th European Simulation Multiconference. Prague, June 2001. Edited by Eugene Kerkhoffs and Miroslav Snorek. Published by The Society for Computer Simulation International, San Diego, USA, pp. 154-160.

Dahl Ole-Johan (1970) Programming Languages as Tools for the Formulations of Concepts: Proceedings of the 15th Scandinavian Congress [Oslo 1968]. Published by Springer, Berlin, pp. 18-28.

Dahl Ole-Johan, Myhrhaug Bjørn and Nygaard Kristen (1968). Common Base Language. Norsk Regnesentralen, Oslo. 2nd edition 1972, 3rd edition 1982, 4th edition 1984.

Dahl Ole-Johan and Nygaard Kristen (1968) Class and Subclass Declarations: Simulation Programming Languages. Edited by John N. Buxton. Published by North Holland, Amsterdam, pp. 158-174.

Geoffrey Gordon (1961) A General Purpose Systems Program: Proc. 1961 EJCC. Published by MacMillan, New York, pp. 81-98.

Geoffrey Gordon (1969) System Simulation. Prentice Hall, Englewood Clifs.

Kindler Eugene (1994) Reflective Simulation in SIMULA. ASU Newsletter, Vol. 22, No. 1, pp. 1-14.

Kindler Eugene (1995) Simulation of Systems Containing Simulating Elements: Modelling and Simulation 1995, Proceedings of the 1995 European Simulation Multiconference. Edited by Miroslav Snorek, Milan Sujansky and Alexander

Verbraeck (eds.). Published by The Society for Computer Simulation International, San Diego, pp. 609-613.

Kindler Eugene (1998) Transplantation – What Causes it in MS-DOS SIMULA?: *Object Oriented Modelling and Simulation of Environmental, Human and Technical Systems – Proceedings of the 24th Conference of the ASU, Salzau (Schleswig Holstein, Germany).* Edited by Broder Breckling and Henry Islo. Published by Ecology Center, Kiel, pp. 155-164

Kindler Eugene (2000) Nesting Simulation of a Container Terminal Operating With its own Simulation Model. JORBEL (Belgian Journal of Operations Research, Statistics and Computer Sciences), Vol. 40, No. 3-4, pp. 169-181

Kindler Eugene (2001) Computer Models of Systems Containing Simulating Elements: CASYS 2000 – Fourth International Conference. Edited by Daniel M. Dubois. Published by The American Institute of Physics, Melville, New York, AIP Conference Proceedings 573, pp. 390-399

Kindler Eugene, Krivy Ivan and Tanguy Alain (2001a) Tentative de Simulation Réflective des Systèmes de Production et Logistiques: MOSIM'01 – Actes de la troisième conférence francophone de MOdélisation et SIMulation. Edited by Alexander Dolgui and François Vernadat. Published by The Society for Computer Simulation International, San Diego – Erlangen – Ghent – Delft, Vol. 1, pp. 427-434

Kindler Eugene, Krivy Ivan and Tanguy Alain (2001b) Reflective Simulation of Discrete Logistic and Production Systems: Modelling and Simulation 2001, 15th European Simulation Multiconference ESM2001. Edited by Eugene Kerkhoffs and Miroslav Snorek. Published by The Society for Computer International, Delft, pp. 861-865.

Kindler Eugene, Krivy Ivan and Tanguy Alain (2002a) Towards Automatic Generating of Reflective Simulation Models: MOSIS '02 – Proceedings of 36th Spring International Conference Modelling and Simulation of Systems. Edited by Jan Stefan. Published by MARQ, Ostrava, Czech Republic, pp. 13-19

Kindler Eugene, Krivy Ivan and Tanguy Alain (2002b) Reflective Simulation of Logistic and Production Systems: HMS/MAS 2002 – International Workshop on Harbour, Maritime and Multimodal Logistics Modelling & Simulation, Modelling & Applied Simulation. Edited by Agostino G. Bruzzone, Yuri Merkuryev and Roberto Mosca. Published by DIP – Genoa University, Liophant Simulation Club, and McLeod Institute of Simulation Science – Genoa Center, pp. 97-102

Krivy Ivan, Kindler Eugene and Tanguy Alain (2002) Software for Simulation of Anticipatory Production Systems. International Journal of Computing Anticipatory Systems, Vol. 11, 2002, pp. 320-335

Madsen Ole L., Møller-Pedersen Birger, and Nygaard Kristen (1993) Object-Oriented Programming in the Beta Programming Language. Addison Wesley, Harlow – Reading – Menlo Park

Novak Petr (2000) Reflective Simulation with Simula and Java: Simulation und Visualisation 2000. Published by The Society for Computer Simulation International, European Publishing House, Ghent, pp. 183-196

Rescigno Aldo and Segre Giorgio (1961) La Cinetica dei Farmaci e dei Traccianti Radioattivi (Kinetics of Drugs and of Radioactive Tracers – in Italian), Boringheri, Torino

Sheppard Charles W. (1962) Basic Principles of the Trace Method – Introduction of Mathematical Tracer Kinetics. Wiley, New York, London

SIMULA Standard (1989) Simula a.s., Oslo

Vrba Pavel (1999) Programming Language Beta – a Tool for System Analysis?: MOSIS'99 – Proceedings of 33rd Spring International Conference – Modelling and Simulation of Systems. Edited by Jan Stefan. Published by MARQ, Ostrava, Czech Republic, pp. 141-156

Vrba Pavel (2000) Systémové aspekty objektove orientovaného přístupu (System Aspects of Object-Oriented Approach – in Czech). Doctoral Thesis, University of West Bohemia, Pilsen, Czech Republique