Models of Aristotelian Concepts in Computer Programming

Eugene Kindler
Department of Mathematics, Faculty of Sciences,
Ostrava University, CZ – 701 03 Ostrava 1, 30. dubna 22
Fax: +420-2-2191-4323; e-mail: KINDLER@KSI.MS.MFF.CUNI.CZ

Abstract
The paper considers some analogies between Aristotelian concepts and their images at the computer programming, applicable for computer modeling of anticipatory systems. The concepts are namely matter/form and their influence to the law of extensionality, general/individual and time/eternity.
Keywords: Traditional logic, Philosophy of mass, Object-oriented programming, Time, Computers

1 Introduction

There is a large group of material systems made or controlled by humans. Let they be simply called man-made systems. The production/logistic systems, those of health care, of services, of agricultural production, etc. can serve as examples of them, but a lot of objects have similar system properties although they might seem not to have too many man-made components (systems like woods, rivers, humans or animals under a strong individual medical procedures, etc.). An essential property that characterizes the man-made systems is that they are consciously controlled by humans or governed by automata constructed by a conscious work of humans. Often a certain professional level of the control is desired: the controlling humans should form their decisions so that they respect the instantaneous situation existing in the controlled systems and derive the decisions from what they observe, applying the rules of causality as much as possible. The controlling automata should model what humans would make if they were at their places. In other words, the controlling automata model the controlling humans and contribute by their much greater capacity of storing information, by their much greater rate of deriving consequences and by their ability to form much greater chains and trees of logical consequences. The mentioned quantitative differences between humans and automata (computers) can grow up into qualitative differences, but with the exception of rather rare cases the present practice neglects this phenomenon and considers the automata as quantitatively powerful "human idiots". It makes possible to neglect some special aspects of the controlling automata and to formulate the next argumentation only for the case when humans control the systems. Let they be simply called *dispatchers*.

The man-made systems belong to the physical world and their dispatchers have to recognize the properties of the material elements of the systems they control. They must often detect phenomena for which a description in standard professional languages of (natural) sciences is too fine, too complex for being tested and therefore useless for deriving logical consequences and formulating decisions. As an example we can present phenomena "fault of a machine", "length of a queue" or "age of a patient". When they are to be detected the dispatchers do not use methods of physics or biology, but they recognize them similarly as the humans recognized all the phenomena of matter before the sciences existed or outside sciences (e.g. the age of a patient can be determined by a simple looking into a form or into a database, a fault of a machine can be recognized by watching a certain indicator). The ideas of the first philosophers and namely the successful general results of the philosophy of matter can be applied to the dispatchers' observation in the same manner as they were applied to the ancient Egyptian architects, to the old Greek businessmen with wine, to the Phoenician sailors, to the Roman architects or to the medieval Cistercian "water engineers". The dispatchers often derive the consequences of their observation by the same logic as the "habitual" humans, i.e. they (often unconsciously) use the logic the rules of which were often formulated by the philosophers who expressed the common laws concerning the matter. Note that the last statement doesn't introduce some antagonism between the logic of the early philosophers and that of contemporary scientists but would like to express that different logical systems neglect certain aspects that are essential in thinking on the material world (e.g. mathematical logic respects axiom of extensionality and essential contents of the relation general-individual).

The dispatchers' often base their decisions on their anticipatory processes. Such processes are bound with the material components of the system, which they concern and use the same logic as mentioned above. The evident consequence of it is to include results obtained by ancient philosophers into the research and practice of the anticipatory systems. Really, we have met the first results in the proceedings of CASYS conferences – see e.g. the classification of causes formulated by Aristotle and given into relation of the system theory by Dubois (2000). From another side, recommending the people living in the modern epoch facilitating by physics, chemistry, mathematics, mathematical logic and other modern sciences to return to the results obtained more than 2000 years ago may seem strange. The present paper should support respecting and accepting the way of ancient philosophers; the reason is that the contemporary programming of digital computers offers a surprisingly rich analogies with general statements on the world and universe, formulated by old authors in the ancient times and Middle Ages and offer new insights to the old results and problems of philosophy.

2 Situation in Computer Programming

Although modern computing technique has existed since a short time of about 50 years, its development surprises by its speed and spreading and one expects it to continue in future with a similar dynamics. Modern computing technique is an important and sophisticated component of communication and of information storing, retrieval and processing (where the term "information" should be interpreted not only as data but in a more general way – e.g. as relations, models, knowledge and concepts). And even when one is conform to the fact that modern computing technique will never replace the human thinking in the complete manner, he has to take into account the historical phenomenon that modern computing technique accepts a lot of properties of thinking and becomes a certain model of it (in sense of an amplifier or of a prosthesis, not of a pattern, prototype or archetype). In that manner, interpretation of the human's application of computing technique turns to be a certain model of philosophy, namely of that in its initial stage in the ancient Greek epoch, when the philosophers were stimulated not only by questions of one's own existence but also by those of the practice of the everyday communications between the humans. In other words, the computer science as it exists nowadays gives new views at a lot of philosophical concepts, attitudes and positions that were often grown and formulated already in the epoch of classical Greek civilization and that were able to be used also outside "pure philosophy" (for the everyday communication, namely in questions and problems connected with the law, business and/or moral areas, for foundations of sciences etc.).

The development of the computer modeling and of the knowledge representation at computers is very fast and nothing shows that the dynamics of this development should be limited in the near future. According to that expectation an analogous expectation can be formulated: while we observe that nowadays there are interesting and surprising analogies between results of the ancient philosophy and the practice of computer programming, we can expect in the future these analogies to grow in their number and in their internal contents as well. Some use of computer recalls a thinking of a rather limited specialist today – e.g. the use of the language GPSS (Schriber 1991) but it develops so that it will more and more recall a larger professional thinking of adults (Kindler and Islo, 1995). Although the results of ancient philosophers, mentioned in the present paper is partially applicable to a thinking of such limited specialists, their fully interpretation excels in relation to the thinking of the persons with a large horizons.

The anticipatory activities were very important stimuli for the development of the ancient philosophy: a lot of such activities were made in order to know the medical, economical, military, agricultural and existential aspects expecting individuals, communities or human kind as a whole. Among the applications existing outside the pure philosophy, such activities took place, though the pure philosophy tended also to formulate more or less general principles that could be today related to the theory of anticipatory systems – we have already mentioned the interpretation of Aristotle's classification of the causes in the system theory and the choice of the conceptions of Aristotelian logic presented in CASYS conferences.

But the present-day computer science fights with some problems that are very similar to those, with which the ancient thinkers fought, and thus some discoveries arise that are surprisingly similar to those made more than 2000 ago. A certain analogy of the development in the ancient philosophy will surely pass inside the computer science, and during it the following aspects are to be expected:

1) The analogy in computer science will be given by the demands rooting in the applications of the computing technique and not by some l'art-pour'l'artism of intellectuals, who might try to give the computing technique the direction similar to that of the ancient philosophy.

2) The development of the computer science will present more and more analogies with that of the philosophy but never will be able to replace it; it will never be able to formulate such notions as that of existence, God, the good, love, sin etc. in an exhausting form but it will be able to make some models and illustrations of them or of their interesting properties (in other words: it is not realistic to expect the computing technique to replace good philosophers but it is to expect it to help them and their pupils more than by serving by databases, vocabularies, registers and partially translating software, as it is able to serve nowadays). The modeling of anticipation activities will be one of the basic stimuli for such a development.

What was written until this line may seem being as a rather abstract, academic and general vision. Nevertheless the contemporary development of the computer modeling offers several results that can serve as certain ways to concretize the expressed ideas.

3 Matter and Form in Case of Storage Mediums

Matter/form (materia/forma, 'ὑλη/μορφή) is a basic pair of concepts of the philosophy of mass. The pair was profoundly studied in relation of physics, especially to modern physics. Nevertheless the computer science offers a new insight into the matter-form relation, namely by means of a similar relation between a part of memory medium of a computer and its contents. The abstraction from the memory media during the programming and modeling work neglects any properties of the physical matter, at which information is stored and considers it only as something formless that is only able to store information. Let us illustrate it at an example.

In the applications of every algorithmic programming language *variables* are introduced. When one introduces a variable he knows that it is automatically represented as something coded at a piece of the computer storage medium (PCSM) but neglects that. More exactly said, one uses a variable *X* as a name for both a PCSM *M(X)* and for an information *J(X)* stored at it but he respects only *J(X)*, he may take into account that there is something like *M(X)* that carries *J(X)* but he neglects all other properties of *M(X)*. For the user of the programming language, the PCSM behaves as a *materia prima* and the information stored at it as its *substantial form*.

More illustrative examples are offered by the programming languages that permit their users to use them to reflect *objects* (more material and concrete entities than variables). They are e.g. the discrete event simulation languages or object-oriented languages. As an example, we can present a model of a *car* (train, airplane, human, lathe, building, field, factory, portion of metal, portion of water, ...) in a simulation model. Such a model of a car is a PCSM carrying all properties of the modeled car, which are taken in account in the modeling process. If the properties change the PCSM does not change; the PCSM fully corresponds to the *materia prima* of a real car *R* (for which *C* is its computer model) and the information coded at the PCSM corresponds to the substantial form of *R*. Note that to think about the PCSM would be against the reason of the applied programming language; the programming languages that permit their users to handle the PCSM (for example to give him the address of PCSM) are considered as “hybrid” (e.g. mixing the object-orientation with autocode programming, i.e. not perfect). They allow the programmers to make errors that end with “wild run”, i.e. with a collapse of the computer operating system used. And they enable to make errors that could be illustrated like “take a part of the materia prima of a car and add that portion to the materia prima of something completely different from the car, e.g. to a distant road. Note that different objects have always different (and mutually disjoint) PCSMs.

The analogy between matter/form and PCSM/stored information is supported by the practical use of the programming languages for computer modeling: their essential contribution is that they lead their users to describe the modeled objects instead what should happen in the computer. Their users should be made free from the complicated and difficult transformation between the description of the modeled reality and that of the corresponding processes inside the computers (the transformation is made automatically by the computer). So the organization of the information stored inside the computer should follow a lot of relations existing at the modeled reality. It is just the effort to make the expressing tools of both the sides of the modeling (the real things and their models inside computers) as near as possible, that promises in the future development of the modeling technology to move the view to the programming concepts even nearer to the classical philosophy of mass than as it is nowadays.

4 Axiom of Extensionality

The analogy between the matter and the PCSM is a base for viewing on the absence of the *axiom of extensionality*, i.e. of the negation of the statement “if the sets have the same elements they are equal”. This axiom is fundamental for any thinking inside mathematics, but fails outside it. For example, in mathematics only one empty set exists and when two two-dimensional points (i.e. ordered pairs) have the same coordinates they are viewed as one point (although the old geometricians have introduced that implication as a certain definition of the equality among the points, in the later time of mathematics it appeared to follow from the axiom of extensionality). Outside mathematics, one could observe e.g. two empty bags so that the bags themselves were not considered to be important, and those bags were understood as different, because one of them could be made non-empty by putting a thing into it, while the other remained empty (note that a similar way of thinking can be met in computer animation – two point having the same place at the computer display can be considered as different points, because they can change their positions to be seen at different places). While the absence of that axiom outside mathematics and computer science can be explained by the existence of the matter the foundation of the absence in the computer science consists just in the memory medium. As an evident illustration, we can present a pair of cars that have the same parameters: in the real world they differ by their portions of matter (these portions are disjoint), while if modeled at a computer they differ by their mutually disjoint PCSM – note that both PCSM can carry the same information (both the modeled cars would have the same properties).

But we can present a more illustrative example. Let *P*=[*x*,*y*,*z*] and *Q*=[*u*,*v*,*w*] be two points. In accord with the axiom of extensionality, it is commonly accepted by the science of geometry that in case *x*=*u*, *y*=*v* and *z*=*w* there are no two points but one point, *P*=*Q*. Nevertheless, the object-oriented programming permits to model two different points with the same coordinates (note that in geometry the coordinates cover the substantial form of the points). Let us present several examples, written in the programming language SIMULA designed as the first object-oriented language by Dahl, Myhrhaug and Nygaard (1968) and used at many computers (beside others workstations, Macintosh and IBM PC compatible (SIMULA 1989)):

```
class point(x,y,z); real x,y,z;
begin real procedure distance_from(P); ref(point)P; distance_from:=
        sqrt((x-P.x)**2+(y-P.y)**2+(z-P.z)**2);
    ref(point)procedure projection_to(j); real j; projection_to:=new point (x,y,j);
    procedure move_to_right_of(j); real j; x:=x+j;
    real module;
    module:=sqrt(x**2+y**2+z**2)
end of point;

class horizontal_circle(center,radius); ref(point)center; real radius;
begin Boolean procedure contains(P); ref(point)P; contains:=
        z=P.z and then center.distance_from(P)=radius;
    procedure move_to_right_of(j); real j; center.move_to_rihgt_of(j);
    real surface, length;
    surface:=3.14159*radius**2; length:=1.5708*radius;
end of horizontal_circle;

horizontal_circle class vertical_cylinder(height); real height;
begin ref(horizontal_circle) procedure lower_base; lower_base:-
        this vertical_cylinder qua horizontal_circle;
    Boolean procedure contains_inside(P); ref(point)P; contains_inside:=
        P.z > center.z and then P.z < center.z+height and then
        lower_base.contains(P.projection_to(z));
    real volume,surface;
    surface:=lower_base.length*height+lower_base.surface*2;
    volume:= lower_base.surface*height
end;
```

The text *horizontal_circle* **class** *vertical_cylinder* states something like "every vertical cylinder is a horizontal circle", in a similar sense like "every *dog* is an *animal*", i.e. dog is a sort of *animal*). The concept *horizontal_circle* has a greater extent and a lesser contents than the concept *vertical_cylinder* (it is possible that horizontal circles exist that are not vertical cylinders, but the horizontal circles hat are not vertical cylinders have no height and volume and it is not meaningful to speak on their lower base). The vertical cylinder is understood as a "thick horizontal circle".

The "attributes" of the classes are values introduced for them; e.g. *horizontal_circle* has attributes *center*, *radius*, *surface* and *length*. The "methods" of the classes are algorithms introduced for the classes; e.g. *move_to* and *contains* are methods are methods of class *horizontal_circle*. A class "inherits" attributes and methods of its "prefix", i.e. of the class the name of which is possibly written before the key word **class**. So the class *vertical_cylinder*, having attributes *height*, *volume* and *surface* and methods *contains_inside* and *lower_base*, "inherits" also all attributes and methods from class *horizontal_cylinder*. The inheritance of the attributes and methods is an essential tool for modeling the subordination of concepts. As it will be shown, SIMULA reacts to the context, and therefore there is no problem that class *horizontal_cylinder* has two attributes having the same name *surface* – one introduced directly for that class and the other inheriting from its prefix *vertical_circle*. The attributes and methods can be used by the help of "dot notation": if *A* is a name of a vertical cylinder and *E* is a name of a point then *A.volume* gives the volume of *A* and *A.contains_inside(E)* gives an answer whether *E* is inside *A*.

Note that the examples presented in this paper are very simple, nevertheless they admit to handle for example two following different points *P* and *Q* that are identical in the geometrical sense, five following different circles *A*, *B*, *C*, *D* and *E* that are identical in the geometrical sense and eight following vertical cylinders *F, G, H, I, J, K, L* and *M* that are also identical in the geometrical sense:

```
P:-new point(4,3,6); Q:-new point(4,3,6);
A:-new horizontal_circle(P,10); B:-new vertical_cylinder(P,10);
C:-new horizontal_circle(Q,10); D:-new vertical_cylinder(Q,10);
E:-new horizontal_circle(new point(4,3,6),10);
F:-new vertical_cylinder(A,20); G:-new vertical_cylinder(B,20);
H:-new vertical_cylinder(C,20); I:-new vertical_cylinder(D,20);
J:-new vertical_cylinder(E,20);
K:-new vertical_cylinder(new horizontal_circle(P,10),20);
L:-new vertical_cylinder(new horizontal_circle(Q,10),20);
M:-new vertical_cylinder (new horizontal_circle(new point(4,3,6),10),20);
```

A computer scientist knows that *P* differs from *Q* by its PCSM, but the computer users do not need to know that detail – for them an idea is sufficient (unconsciously) to accept that any point has its own portion of matter and that two points always differ by such portions. The same holds on the circles and cylinders (note that matter is not space, though some philosophers declared some strict relations in this sense!).

As an illustration we can present an instruction *K.move_to_right_of(5)*; it is performed in the following steps:

4.1. For the "generalium" *vertical_cylinder* no definition of *move_to_right_of* was defined, but that generalium is a special "sort" of the "genus" *horizontal_circle*.

4.2. For the "generalium" *horizontal_circle* a definition of *move_to_right_of* exists and therefore the cylinder *K* makes it according to that definition (note that *K* makes it as it have been its own lower_base).

4.3. In the mentioned definition it is stated that a horizontal circle makes a move to the right so that it "demands" its center to make such a move; therefore the move of *K* is now delegated to the move of the center of its lower base;

4.4. The center is a point and for the "generalium" *point* the meaning of the words *move_to_right_of* is explicitly defined, namely in a manner different from that belonging to the "generalium" *circle*: the point moves so that it changes its own coordinate. Therefore the move of *K* is interpreted by a change by the coordinate *x* of the center of its lower base.

If we accepted only the conventional geometric viewing, we would have to admit that such a shift caused one cylinder *K*=*L* to became two different cylinders *K* and *L*. If we admit *K* and *L* to be always different cylinders we will be in a more suitable logical situation. The idea of different portions of matter makes it possible and the different PCSMs make possible to model it at computers. Note that in anticipating models we often meet objects that are the same in a mathematical sense but later each of them can develop differently from the others.

5 Nominalism Versus Realism

Some philosophers of the our days and a lot of scientists consider the controversies between nominalists and realists as a historical phenomenon that concerns the Middle Ages and is not interesting for the contemporary scholars more. Nevertheless it is just the mentioned object-oriented programming that carries an unconventional method of knowledge representation giving a new insight into that aged philosophical confrontation: the object-oriented programming handles with the "generalia" so that it allows to analyze and construct their connections based on the hierarchy of relations "lesser extent and greater content" and to make "instances" (individual entities) of those "generalia" so that such instances completely reflect the relations introduced for the "generalia" and mutually differ both by their formal properties and by their matter (PCSMs).

A good illustration that the computer can model "generalia" and relation among them was presented in the example in the preceding section, namely in the logical steps for performing the procedure *move_to_right_of*. If for example the "generalium" *vertical_cylinder* were not introduced as a "sort" of another generalium *horizontal_circle*, *K* would not be able to react to the instruction of moving, because it could not meet any definition of it. From the other point of view, we can observe that the "generalium" *vertical_cylinder* has two "attributes" called *surface*: one was defined directly for that generalium (in class *vertical_cylinder*) while the other one is "inherited" from the generalium (introduced in class *horizontal_circle*). The first corresponds to the concept of surface of a cylinder while the last corresponds to the concept of surface of a circle, i.e. – for the cylinders – for their (lower) bases. The language SIMULA offers several tools to express whether *K* performs something as a cylinder or as a circle (see the line **this** *vertical_cylinder* **qua** *horizontal_circle* of the definition of lower_base above – one can write *K* **qua** *vertical_cylinder* to express that *K* performs something as a cylinder, and one can write *K* **qua** *horizontal_circle* to express that *K* performs something as a circle; the programmers understand **qua** as an abbreviation of "qualified as", but **qua** can be interpreted as a Latin word and understood as the English word "as").

The models of the generalia can exist and be applied independently of humans who expressed them: I can represent *vertical_cylinder* at a computer and forget it while another person can used it a long time after. It is interesting that the importance of the old philosophical ideas of realism and nominalism appeared after (unsuccessful) struggling with the problems that had to make clear questions in communication belonging to the typical everyday programming work. The questions of the computer business could seem to be much more simple than the profound philosophical formulations of medieval scholars. But the development of the technique of computer representation of knowledge and its application tends to the formulation and manipulation with much more complex "generalia" and so to reduce the distance between the sorts of questions of programmers and those of philosophers. Then the programming practice will be able to model and illustrate some contradictions between radical realists, Thomists and nominalists. Already nowadays it is possible to represent e.g. an idea of imagining specialist (Kindler, 2000). Nevertheless the relation between the philosophical attempt and computer modelers' view, as it exists nowadays, is a clear illustration that the communication between realists and nominalists is more modern subject than it is generally assumed.

Note that among the authors of books and papers about the object-oriented programming there are similar confusions and contradictions as among the medieval philosophers discussing the size of realism. For example, authors do not differ between instantiation of a class (forming an object, e.g. an individual cylinder, in SIMULA by using the key word **new** – see above) and defining a hierarchy among generalia. Or some authors present class as a certain instance that is a "prototype" for the other instances of the class – one can watch an analogy between the classes-prototypes and Plato's ideas.

6 Computer Simulation

In the Aristotle's heredity there is also the philosophy of time. We can say that it is a philosophy of Newtonian time (i.e. of a time that flows without relativistic effects), but with neglecting the fact that time moments can form infinite ordered sets. The same viewing on time exists in *discrete event simulation*.

Computer simulation is a special way of computer modeling. In a rather simple manner, it can be described as an application of a program (called *simulation program*), which turns the applied computer to an analogy of a certain process passing in the Newtonian time. That process is called *simulated system*. It is possible to say that the running computer controlled by a simulation program behaves similarly as the simulated system, namely so that the order of the "events" in the simulated system (i.e. short steps forming its development during time) is the same as the order of their representations in the computer run. More about computer simulation can be read e.g. in (Dahl, 1964) or in (Genuys 1968). It could be said that a good simulation program generates information on the simulated system of a quality so good that a person who does not know its origin can believe that it to have been measured and detected at a real system. Therefore computer simulation enables to study a possible behavior of a system that does not exists (e.g. that is designed) or a possible behavior of an existing system after performing some decision in it.

Computer simulation exists in three sorts: *discrete event simulation* models the systems in which the Aristotelian view to time is respected; *continuous system simulation* models systems in which the Aristotelian view is completed by respecting the continuous time flow and completed by Newtonian consequences oriented to the interpretation of time derivatives; and *combined discrete event continuous system simulation* (shortly *combined simulation*) that permits to include intervals of Newtonian time into the Aristotelian one. The continuous system simulation and the combined one are too bound by the processes described by (ordinary) differential equations and so they are distant from the problems and results of the Aristotelian philosophy and also limited in their world viewing. Therefore we will concentrate our attention to the discrete event simulation, which we will in the future text call simply simulation.

Simulation was successfully used for anticipating the future behavior of a great number of production, logistic, environmental, health care, military, energy, machine, electronic, chemical and service systems and of organisms and social groups. But the simulation programs can be also applied for training and for illustrating situations that would be dangerous and/or expansive: such a situation is simulated at a computer and one can watch them and their reaction to one's interventions. Simulation was not used for some deeper psychological or philosophical processes. But it can be applied in a more sophisticated manner, as it will be described in the following two sections. They give further illustrations of the view to time formulated by the old philosophers.

7 Simulation of Worlds With Different Times and Physics

In a professional terminology of simulation, a computer run described in the preceding section is called *simulation experiment*. During it, the time in which the computing really exists maps the time of the simulated system so that if an event E_1 comes in the simulated system before an event E_2 the "image" of E_2 in the computer run cannot come before the "image" of E_1 (the image of an event is also an event, but inside the computer). In other words, during a simulation experiment the simulated time cannot flow in the direction opposite to that of the time of the simulated system. Note that this ordering is of a great practical importance: it leads the authors of simulation programs to reflect possible causal dependences among the events existing at the simulated system in the simulated program.

But the simulation experiments are often repeated to get some more complex information, e.g. to decide the optimal variant of a system that has to be installed, or the optimal variant of an intervention into an existing system. Such a sequence of simulation experiments is called *simulation study*. At the beginning of each simulation experiment of a simulation study, the simulated system is reflected as being newly constructed; it concerns the simulated time, too. Therefore a simulation study looks like a computer model of a certain environment (let us call them "eternity") which has no (Newtonian) time and in which there is somebody who is able to create "physical worlds" with their own times and mutually independent in the manner that they cannot interact by means of their own properties and instruments: the only interaction between the physical worlds is that they "export" information about themselves to the eternity and there the "creator" can use them for an intervention to the next physical worlds. Note that the simulated worlds can mutually differ not only by their times but also by any of their parameters, control laws and even by their physical and/or social laws, by their professional concepts etc. A statement expressed by St. Thomas Aquinas in Summa contra Gentiles may be referenced in this context – "not in time but with time the God created the universe".

8 Parallel Existence of Simulation Models

In a conventional simulation study the simulation experiments alternate: the extinction of a simulation experiment permits to generate the next simulation experiment. At the used computer, more than one simulation experiment cannot exist contemporary. One speaks on *serial simulation study* in that case.

Nevertheless there are some simulation studies, in which more than one simulation experiments exist at the same time. They can be called *serio-parallel* or simply *parallel simulation studies*. Such studies are rather rare because to implement them is very difficult; nevertheless the Czech simulationists have good experiences in that branch and their results are worldwide appreciated and applied (see e.g. section 2.1 of (Kindler 2002)). They model an "eternity" in which any physical world can rise, exist and disappear independently of the existence of other. Each of the physical worlds has its own time and so several independent times can exist "at the same time" in the same eternity. The words "at the same time" are written in quotes, because there is a certain contradiction – the eternity need not have its own time. Nevertheless, the computer model maps some phenomena that are under the control of a certain ordering and that can stimulate to think on certain events in which the eternity encounters a physical world (note that the ordering of the events in the eternity need not to imply a concept of distance between them – the events form an ordered set but do not need to form a one-dimensional linear metric space).

9 Conclusion

For a philosopher who well knows the results of the ancient philosophy, the analogies between the profound philosophical results and the mentioned concepts of the computer programming may seem to be at the very surface of the ontology, logic and noetics. The corresponding concepts of the computer practice may seem to touch the very façade of philosophy. Nevertheless, let us note that the development of computer programming exists for about 50 years and during this period it developed rather dynamically: in the

50ies the computer programming was a technology with chaotic relations to the human thinking (the real bindings between the programmer's intentions and the corresponding results of his work were considered as private and internal aspects of the programmer's intellectual life), since that time it came to the state of touching the "façade" of real philosophy and in the next decades it is expected to approach to the human thinking as much as possible – that will not be a demand of philosophers but that of the programming praxis (a demand to shorten the transfer and translation from the human thinking to the computer operation). Therefore the computer programming is expected to penetrate more behind the façade of philosophy. Some modern aspects are already certain illustrations and stimuli of the future development, like object-oriented knowledge systems containing peaces of knowledge that are themselves object/oriented knowledge systems (or – in another viewing – formal theories, among the entities of which there are other formal theories). Nevertheless, nowadays such stimuli are not elaborated in their greater plentitude and so they could not be included into the present paper.

References

Dahl Ole-Johan (1964) Discrete Event Simulation Languages. Norwegian Computing Center, Oslo. Reprinted in (Genuys, 1968).

Dahl Ole-Johan, Myhrhaug Bjorn, and Nygaard Kristen (1968) SIMULA Common Base Language (1st ed.). Norsk Regnesentralen, Oslo, 1972 (2nd ed.), 1982 (3rd ed.), 1984 (4th ed.).

Dubois Daniel M. (2000) Review of Incursive, Hyperincursive and Anticipatory Systems – Foundation of Anticipation in Electromagnetism. Computing Anticipatory Systems: CASYS'99 - Third International Conference. Edited by Daniel M. Dubois, Published by The American Institute of Physics, AIP Conference Proceedings 517, pp. 3-30.

Genuys Fernand (editor) (1968) Programming Languages. Academic Press, London – New York.

Kindler Eugene (2000), Simulation of systems with imagining components. Workshop 2000 – Agent-Based Simulation. Edited by Urban Christian, Published by The Society for Computer Simulation – Europe, pp. 239-244.

Kindler Eugene (2002) When Everybody Anticipates in a Different Way .... Computing Anticipatory Systems: CASYS'99 - Third International Conference. Edited by Daniel M. Dubois, Published by The American Institute of Physics, in print.

Kindler Eugene and Islo Henry E. (1995) Independent Object Components and Textual Enclosures. ASU Newsletter, vol. 22, no. 3, pp. 4-13.

Schriber Thomas J. (1991) An Introduction to Simulation using GPSS/H, Wiley, New York.

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