40 YEARS OF OBJECT-ORIENTED AGENTS

Eugene.Kindler
Department of Informatics and Computers
Ostrava University, Faculty of Science
CZ-170 00 Prague 7, Czech Republic
E-mail: ekindler@centrum.cz

KEYWORDS

Object-oriented programming, Simulating agents, Simula, Intelligent agents

ABSTRACT

In 1967, SIMULA 67 was presented at the IFIP Working Conference on Simulation Programming Languages. Although the exact definition of SIMULA 67 was a bit modified during the next year, the essential properties of the first proposal were conserved and respected. Among them, the properties declared later as those characterizing the object-oriented programming (classes, subclasses and virtuality of methods) were introduced, but also other properties like (a) "life rules" of running in quasi-parallel systems at a mono-processor computers, and (b) classes local in blocks and in objects. Although property (a) rooted in an old practice introduced for discrete event simulation, it was ignored in many object-oriented tools, but – combined with the object orientation - it leads to agents. Property (b) leads to combining simulating agents reflected in a simulation models that could be programmed by different agents. Applications will be presented, too.

FROM PROCESSES TO AGENTS

Programming simulation models is a difficult task. Simulation programming languages tried to help it in the following way. Instead of describing what should happen in the computer, the author of the model describes the simulated system and the description is automatically converted into the form acceptable by the applied computer. One of the famous languages for discrete event simulation was GPSS, applied since the early sixties until nowadays (Gordon 1961, Schriber 1974, Schriber 1991). It reflected the fact that the systems are often composed of elements (called *transactions*) that behave according to certain rules (algorithms, may be called *life rules*) and in that behavior interact one with others. The life rules make the transactions to be *initiative* like the later true agents. The transactions can enter the system and leave it. Similar transactions are instances of their common class and the life rules are connected with the classes.

The essential help consists in automatic switching among performing algorithms (life rules) carried out by different transactions, which causes an illusion of their contemporary behavior and dynamics. The tools serving to it referred to the modeled time and were covered by term *scheduling statements*. Although the GPSS tools for algorithmizing the life rules and for interaction between the transaction were rather poor, the decomposition of the whole system dynamics into the transactions caused so vehement feeling of their autonomy that even the specialists in agent paradigm (Urban 2000) did not hesitate to view GPSS applications as those of agents (Florea and Kalisz 2000).

The development of similar languages continued, namely by improving the tools for interactions between the transactions and the algorithmic tools for describing the life rules. Instead of transactions, one spoke on parallel *processes* (Dahl 1966). The most perfect fruit of this development was SIMULA (Dahl and Nygaard 1965 and 1966). This language assumed the full algorithmic apparatus of Algol 60 (Backus et al. 1960), including the true block structure, and allowed the use of scheduling statements in the blocks marked by the users as simulation models. For the interaction between the processes, Simula offered a *connection statement* (called often *inspection*): in its life rules, any process *P* could meet a statement of a form i*nspect Q do S*, where *Q* is another process and *S* is interpreted according to the same manner as the life rules of *Q*.

FROM PROCESSES TO OBJECT-ORIENTED PROGRAMMING

In 1966, one of the authors of SIMULA, O.-J. Dahl, was invited to be a lecturer at NATO summer school on programming languages in Villard-de-Lans (France), where he met another lecturer C. A. R. Hoare and his lecture published later as (Hoare 1968). Hoare introduced an idea of hierarchical classes of data and a manner of their referencing called *remote identifying* and later *dot notation*. The hierarchy of data classes existed so that a class (1) introduced data structures with certain components, and (2) could be "specialized" to its *subclass* that assumed other components. The dot notation told, that if *R* was a reference to a data structure having an item called *W* among its components, then *R.W* referred to "*W* of *R*". If *W* itself was a reference to a data structure that had an item called *X* among its components then *R.W.X* referred to "*X* of *S* where *S* was *W* of *R*".

Proceedings 21st European Conference on Modelling and Simulation
Ivan Zelinka, Zuzana Oplatková, Alessandra Orsoni ©ECMS 2007
ISBN 978-0-9553018-2-7 / ISBN 978-0-9553018-3-4 (CD)

O.-J. Dahl often thought back on the Hoare's ideas – e.g. (Dahl 2002) – as on the essential impulse for his way to object orientation. In fact, already the "old" SIMULA had tools for producing an a priori unknown number of instances of classes, so that the instances were richer than those considered by Hoare: they had life rules. Note that such instances were much better facilitated by another aspect later related to agents – cooperation.

Of course, accepting Hoare's idea of class hierarchy into the old SIMULA implied three consequences: (a) the processes could be considered as instances of a specialization of a general class that was independent of simulation and much near to autonome agents, (b) not only the data but also the life rules could be enriched in the subclasses, and (c) omitting the universality of class of processes demands (and offers) a more general way of switching among the life rules: it was introduced under the title *quasi-parallel sequencing*; also that phenomenon put the instances near to the general practice of agent paradigm – the cooperation of the instances was set free from the dependence on modeled time. (a) and (c) turned the language to a general purpose programming language with suitable tools for applying agents and for computer simulation.

Very important (and independent of the Hoare's ideas) was the decision to include procedures into the structures defined by means of the classes, and applying the dot notation to their calling. In general, the statement *X.F(Y)*, meaning "let *X* perform procedure *F* with parameter *Y*" may reflect a natural language phrase with subject *X*, verb *F* and complement *Y*. In relation to the agent paradigm, the same statement, called by agent *Z* can represent a reaction *F* of agent *X* on the state caused by *Z*.

Nevertheless, O.-J. Dahl and his collaborator K. Nygaard had to come to discovering another tool, generally called *virtuality*: the content of a procedure introduced in a class *C* can be declared in a different way in its subclass. Virtuality was generalized also to the targets of the transfers in the life rule algorithms.

Under the term *object-oriented programming* (further OOP), the world professional community considered the paradigm comprising classes and subclasses as encapsulations of data and procedures, and the virtuality. The life rules were not accepted as necessary component of OOP, nor the switching by means of scheduling statements or quasi-parallel sequencing. Their absence (in C++, SmallTalk, late Pascal etc,) embarrassed software preparation for agent application.

TO SUPER-OBJECT-ORIENTED PROGRAMMING

Similarly as the starting simulation language SIMULA, also the new language included full block structuring introduced in ALGOL 60. The authors Dahl and Nygaard realized that class declaration can in general be subject of the same context rules as other declarations (those of variables, attributes, procedures,…) and thus they introduced *local classes*, namely classes local in blocks and in objects. Summed up, the notion of class came to a declaration of data, procedures, life rules and (so called internal) classes. In the subclasses, the virtuality could give new contents to the procedures and to the targets of life rules.

While the class with life rules and procedures became a base for reactive agent, the class containing internal classes became a base for intelligent agents that could use the internal classes as abstract concepts; in other words, an instance of a class handling internal classes can represent an agent that is a model of an intelligent entity that thinks, using general concepts reflected by the internal classes. Or – using yet other words – an instance of a class that has internal classes is an image of an agent using the language introduced by means of the internal classes.

Such an agent can use the internal classes as any other classes, i.e. it can form instances of them and let them operate. It implies that such an agent can operate as a modeling one or even as a simulation one. Also the instances of its internal classes can carry properties of agents, i.e. the agent can be a carrier and organizer of its "private" agents, which does not contradict to the fact that the same agent can cooperate with other agents. Also the blocks with local classes represent communities of cooperating agents, although such blocks themselves are distant from being agents.

The new language was called SIMULA 67 and the original simulation one was renamed as SIMULA I. Nevertheless the users of SIMULA I turned to SIMULA 67 and SIMULA I fell into oblivion. This was the stimulus, that when SIMULA 67 became an international standard referred by ISO in 1986, the complement 67 was omitted, so that nowadays one calls it simply SIMULA. Its OOP tools and properties overpassing OOP, namely life rules, quasi-parallel sequencing and local classes are sometimes characterized by the words super-object-oriented programming (Kindler 2004, SOOP Corner 1993).

MODSIM (Herring 1990) is an object-oriented language with life rules and scheduling statements but without local classes, virtual targets of the transfers in the life rule algorithms, and quasi-parallel systems. Similar properties could be observed at NEDIS (Glushkov et al., 1975). Nowadays, only BETA (Madsen et al. 1993) could be classified as a tool for the super-object-oriented programming and – when we assume with a great broadmindedness Java as an OOP language – we could think on it as on a super-object-oriented tool, too.

NEW RESULTS

SIMULA was widely used in a form satisfying the agent paradigm. But this paradigm itself develops and its tools are being slowly improved. In parallel with that, new tools are implemented in SIMULA in order to make it more suitable for agent practice. The following products should be mentioned.

SIMULA 67 offered a standard tool for discrete event simulation. It was a class called *SIMULATION*, containing an important internal class *process* and enabled the users to prepare the models in a readable and efficient way. Nevertheless, such an application does not allow giving names to the simulation models. The reason consists in security against "transplantation" errors, i.e. against programming errors that transfer an element of a model into another model: in general, both the models can be in different states and such a transferred element can carry inconsistency into the target model. When the models cannot be named the erroneous transfer cannot be expressed.

An analogy to *SIMULATION* was constructed so that it allows giving names to simulation models but is as safe against the transplantation as *SIMULATION*. The new class enables applying a lot of steps typical for agents on models.

Any class containing internal classes enables assembling of their instances into a certain community and can serve for preparing general tools for the mutual communication among them. In many cases it is suitable, but sometimes one prefers to view any instance as an autonomous – in principle separated – agent. SIMULA allows it and nowadays our work consists in preparing classes of such separate agents that could be a posteriori included into any "universe".

SOME APPLICATIONS OF INTELLIGENT AGENTS

The first application of SIMULA where the intelligent agents occurred concern dynamic optimization – it can be characterized as a model of a session of several experts who want to determine the optimum of a discrete event dynamic systems depending on a certain number (in fact one to 30) of parameters; each of them has a computer simulating his variant, during simulation the experts share their experiences on the behavior of their models and according it they modify their variants and – therefore – their simulation models (Weinberger 1987, 1988). The method was applied in metallurgy, machine production, services, project managing and neurology and always appeared surprisingly efficient.

While the mentioned study could be characterized as a non-simulation model of a system of simulating agents, the next example is in a certain sense inverse: it could be characterized as a simulation model containing a model composed of non-simulating agents. It concerned simulation of a rectification column, i.e. of a cascade system behaving according to a complex system of non-linear partial differential equations. For its numerical solution at a digital computer the following special method was applied. It was designed as a certain system of agents that could be seen as models of experts, each of which was using splines to follow the way to the result from his special direction (from the front, from behind, from above, from below and from the past); the agents mutually communicated and modified their data in order to come to a result that would satisfy each of them (Kindler 2002). When the agents have computed a vector of the results in a certain place (temperature, enthalpy, percentages of chemical substances, percentages of liquid and gasiform components) they move together to concentrate to the next place and so they change the whole state of the column. It was repeated during the whole simulation of the rectification column.

Several applications concerned simulation of operation transport in production halls. In the first of them, a hall served by automatically driven induction carriages was simulated so that the carriages were modeled as initiatively computing their shortest paths with respect to the instantaneous trafficability of the transport network segments – in fact some of them could be blocked for a carriage by a barrier caused by another carriage performing there its task (Kindler and Brejcha 1990). In the simulation model of the whole system, the carriages were represented by agents, applying their life rules for moving and working, while their routine for computing the shortest path was implemented as a simulation model of a fictitious system invented by Dijkstra and Lee and described at the end of (Dahl, 1966): it consisted in agents proliferating, spreading and contemporarily moving from the start node along the whole transport network – Dahl in the loc. cit. uses term pulses. For them.

A much more modern application concerned a circular conveyor with rollers connecting working areas and collaborating with a computer that decides on the destination of every transported object, on accepting or releasing an object to the conveyor in case it is rather occupied, and on the continuing with reconfiguring, or immediate repairing in case of a fault (Berruet et al. 2004, Kindler et al. 2004). Simulation of flexible manufacturing systems with automatically guided vehicles belongs to the same sort of application (Tanguy et al. 2004).

A rather similar studies concerned container yards where the operation transport tools (like forklifts) were managed by a central computer that dynamically computed shortest path for each of them, anticipating the possible changes of the network composed of places without containers, at which the transport tools could move (Kindler 1997, 1999).

The intelligent transport with anticipation of possible future states and consequences of the instantaneous decisions is a very efficient stimulus for nesting decompositions into agents – at the upper level the agents correspond to the transporting tools and in the lower level (nested in the agents operating in the upper level) the agents correspond to fictitious cooperating elements that often reflect the images of the agents of the upper level as they exist in the "brain" of the other agents functioning the upper level (see Figure 1 where the models are represented by rectangles with rounded corners and the cooperating agent by the symbol of the moon).

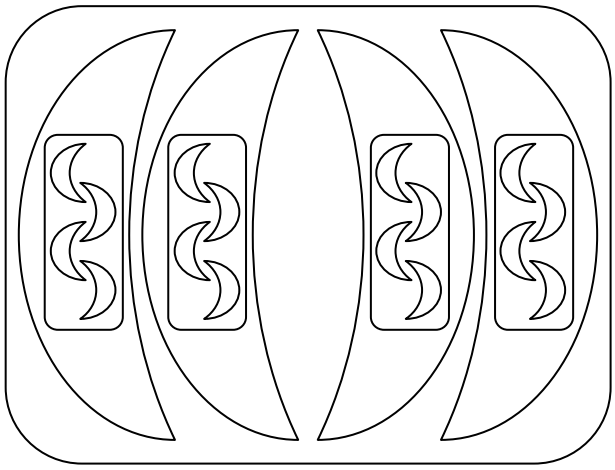

Figure 1: Agents Nesting Inside Other Agents

Such an application concerns simulation approach to a public personal transport in a certain Moravian region where passengers use buses and determine their paths (combinations of consecutive sectors of different bus lines), using imagining various variants of the paths to their targets (Bulava, 2002).

The study just mentioned was accepted into official documents on the region development and was a stimulus for a study that could be characterized as similar for its reflecting human anticipation into the model although it does not concern transport more. It simulates possible demographic development of the same region in future years, including some intelligence of the interested citizens who imagine and anticipate the future changes; the system is intended to be completed by consulting centers for the inhabitants, equipped by simulating computers (Bulava, 2003).

A quite different application can be observed at (Kubeczka 2002), where the "external" agents represent the quantums of transported gas in a gas transport network while the "internal" agents compute how the external agents should operate. The internal agents influence the others in the opposite direction that the external agents do – the external ones reflect the physical move from the inputs to the outputs, while the internal agents reflect the "flow"of the demands (from the outputs to the inputs).

Quite new are initial studies on simulation models of automatically controlled in-patient departments of hospitals (Křivý and Kindler 2005) and on refiguring information systems (kindler, Klimeš and Křivý, 2007).

CONCLUSIONS

What was presented in 1967 as properties of SIMULA 67, in a large measure overpassed what was later called Object-oriented programming. The swicthing life rules, local classes and model nesting present rather distant horizons for the development of intelligent systems modeling and – especially – simulation in the future.

ACKNOWLEDGEMENTS

This work has been supported by the Grant Agency of Czech Republic, grant reference no. 201/060612, name "Computer Simulation of Anticipatory Systems" and by internal grant "Intelligent Information and Knowledge Systems and Their Implementation in Portals".

The autor thanks to his collaborator Profesor Ivan Křivý of Ostrava University for valuable stimuli got during the discussions on the taxonomy of nesting simulation models.

REFERENCES

Backus, J. W. et al. 1960. "Report on the Algorithmic Language ALGOL 60." Numerische Mathematik 2, 106-136.

Berruet, P.; T. Coudert and E. Kindler. 2004. "Conveyors With Rollers as Anticipatory Systems: Their Simulation Models." In: *Computing Anticipatory Systems CASYS 2003 – Sixth International Conference, Liege, Belgium, 11-16 August 2003*, D. M. Dubois (Ed.). American Institute of Physics, Melville, New York, 582-592.

Brejcha, M. and E. Kindler. 1990. "An application of main class nesting – Lee's algorithm." *SIMULA Newsletter* 13, No.3 (Nov.), 24-26.

Bulava, P. 2002. "Transport system in Havirov." In *Proceedings of 28th ASU Conference, Brno, September 26 – October 1, 2002*. Faculty of Information Technologies, University of Technology, Brno, 57-62.

Bulava, P. 2003, "Human-Thinking Simulation". In *Proceedings of the 29th ASU Conference – Object-Oriented Programming in Simulation, Bystřice pod Hostýnem, September 10-12, 2003*, I. Křivý and R. Krpec (Eds.). University of Ostrava, Ostrava, 2003, pp. 19-22.

Dahl, O.-J. 1966. *Discrete Event Simulation Languages*. Norwegian Computing Center, Oslo. Reprinted in (Genuys 1968), 349-395.

Dahl, O.-J. 2002. "The Birth of Object Orientation: the Simula Languages." In: M. Broy and E. Denert (Eds.): *Software Pioneers: Contribution to Software Engineering*. Springer, Berlin, 78-90. Reprinted in (Owe et al. 2004), pp.15-25.

Dahl, O.-J. and K. Nygaard. 1965. *Simula – A Language for Programming and Description of Discrete Event Systems. Introduction and User's Manual*. Norwegian Computing Center, Oslo.

Dahl O.-J. and K. Nygaard. 1966. “Simula – an Algol-based Simulation Language.” *Communications of the ACM* 9, No. 9, 671-678.

Genuys, F. (Ed.). 1968. *Programming Languages*. Academic Press, London – New York.

Glushkov, V. M; V. V. Gusev; T. P. Maryanovich and M. A. Sachnyuk. 1975. *Programmnyje sredstva modelirovaniya nepreryvno-diskretnych sistem (Programming tools for the modeling of continuous-discrete systems – in Russian).* Naukova Dumka, Kiev

Gordon, G. 1961. “A General Purpose Simulation Program.” In *Proc. 1961 EJCC*. MacMillan, New York, 81-98.

Herring, C. 1990. “ModSim: A new object-oriented simulateion language”. *SCS Multiconference on Object-Oriented Simulation*. The Society for Computer Simulation, San Diego

Hoare, C. A. R. 1968. “Record Handling.” In (Genuys 1968), 291-346.

Kalisz, E. and A. M. Florea. 2000. “A GPSS simulation model of interactions in a market-based multi-agent system.” In: (Urban, 2000), 145-150.

Kindler, E. 1997. “Classes for object-oriented Simulation of Container Terminals.” In *Managing and Controlling Growing Harbour Terminals*, E. Blümel (Ed.). The Society for Computer Simulation International, San Diego, Erlangen, Ghent, Budapest, 175-278.

Kindler, E. 1999. “Nested Simulation of Container Yards.” In *Simulation und Visualisierung '99*, O. Deussen, V. Hinz, P. Lorenz (Eds.). Society for Computer Simulation Europe BVBA, Ghent, Belgium, 247-259.

Kindler, E. 2002. “When Everybody Anticipates in a Different Way ...” In *Computing Anticipatory Systems CASYS 2001 – Fifth International Conference, Liege, Belgium, 13-18 August 2001*, D. M. Dubois (Ed.). American Institute of Physics, Melville, New York, 119-127.

Kindler, E. 2004. “SIMULA and Super-Object-Oriented Programming.” In (Owe et al. 2004), 163-182.

Kindler, E.; T. Coudert and P. Berruet. 2004. “Component-Based Simulation for a Reconfiguration Study of Transitic Systems.” *SIMULATION* 80, No. 3 (March), 153-163.

Kindler, E., C. Klimeš and I. Křivý. 2007. “Simulation Study With Deep Block Structuring”. In J. Štefan (Ed.). *MOSIS’07, Proceedings of 41th Spring International Conference Modelling and Simulation of Systems, Rožnov Pod Radhoštěm, April 2007*. MARQ,.Ostrava.

Křivý, I. and E. Kindler. 2005. “Computer Representation of Formalized View of In-Patient Departments of Hospitals.” In *CompSysTech 2005 – Proceedings of the International Conference on Computer Systems and Technologies, Varna, 2005*. Bulgarian Union of Automation and Informatics, Varna, 1-6.

Kubeczka, K. 2002. “Quasi-parallelism in Simulation of Continuous Transport Systems”. In *Proceedings of 28th ASU Conference, Brno, September 26 – October 1, 2002*. Faculty of Information Technologies, University of Technology, Brno, 93-103.

Madsen, O. L.; B. Møller-Pedersen and K. Nygaard. 1993. *Object-Oriented Programming in the Beta Programming Language*. Addison Wesley, Harlow – Reading – Menlo Park.

Owe, O. et al. (Eds.). 2004. *From Obhect-Orientation to Formal Methods. Essays in Memory of Ole-Johan Dahl*. Lecture Notes in Computer Science, 2635, Springer, Berlin

Schriber, T. S. 1974. *Simulation Using GPSS/H*. Wiley, New York – London – Sydney – Toronto.

Schriber, T. S. 1991. *An Introduction to Simulation Using GPSS/H*. Wiley, New York – Chichester – Brisbane – Toronto – Singapore.

SOOP Corner.1993. *ASU Newsletter* 21, No.1 (Febr.), 41

Tanguy, A.; E. Kindler; I. Krivy and P. Lacomme. 2003. “Simulation of FSM Including Automated Guided Vehicle.” In: *Proceedings of the 2003 European Simulation and Modelling Conference “Modelling and Simulation'2003”, Naples, 2003*, B. Di Martino, L. T. Yang and C. Bobeanu (Eds.). EUROSIS-ETI, Ghent, 122-126.

Urban, Ch. (Ed.). 2000. *Agent-Based Simulation. Proceeding of Workshop 2000, Passau, Germany, May 2000*. The Society for Computer Simulation International, San Diego.

Weinberger, J. 1987. “Extremization of Vector Criteria of Simulation Models by Means of Quasi-Parallel Handling.” *Computers and Artificial Intelligence*, 3, 71-79.

Weinberger, J. 1988. “Evolutional Approach to Extremization of Vector Criteria of Simulation Models.“, *Acta Universitatis Carolinae Medica*, 34, 249-258.

EUGENE KINDLER was born in 1935 in Prague (Czechoslovak Republic). He studied mathematics at Charles University in Prague and there he got grades of Doctor of philosophy in Logic and Doctor of sciences in theory of programming. The Czechoslovak academy of sciences granted him the grade of Candidate of sciences in physics/mathematics. During his employment in the Prague Research Institute of Mathematical Machines (1958-1966), he participated at the design of the first Czechoslovak electronic computer and designed and implemented the first Czechoslovak ALGOL compiler for it. Then, working at the Ïnstitute of Biophysics at the Faculty of General Medicine of Charles University (1967-1973), he designed and implemented the first Czechoslovak simulation language and then introduced the object-oriented programming into Czechoslovakia. Nowadays, as professor emeritus of applied mathematics, he teams up with University in Ostrava and with Charles University in Prague. He was visiting professor at the University in Italian Pisa, at the University of South Brittany in French Lorient, at Blaise Pascal University in French Clermont-Ferrand, and at West Virginia University in American Morgantown. His main interest consists in object-oriented and modeling of systems that handle complex models to improve their own anticipatory abilities and intelligence. He applied it namely in logistic and production systems, for example in participating at two grants supported by the European Commission and oriented to modernization of sea harbors by means of computing technique. Beside computer science, he is interested in music. He plays violin, piano and organ and for 30 years he is a director of a singing group performing the Latina, Greek, Armenian and Palaeoslavic chants of the first millennium.