Joint NCC & IIS Bull., Comp. Science, 12 (1999), 42–46

The development of distributed dynamic systems in the technology of active objects

I.E. Shvetsov, T.V. Nesterenko, S.A. Starovit, S. V. Preis

In the paper we present a technology for development of dynamic multi-agent systems which is based on a combination of the object-oriented approach with constraint programming[1]. We introduce the notion of an active object as a way of describing and implementing the intellectual, reactive and communicative properties of agents. We describe computation processes in active objects. A special emphasis is made on the dynamic component of this technology. Dynamic systems are classified with respect to their complexity level. It is shown how such systems can be implemented with the help of the technology of active objects.

Introduction

The concept of an autonomous agent presumes that the agent is functioning on its own in order to solve the problems assigned to it. Thus, each agent possesses certain dynamism manifested in its independent behavior. The architecture of a multi-agent system, however, can be either static or dynamic. When we say "architecture of a multi-agent system," we mean its three main components: types of agents existing in the system, the number of agents of each type, and the set of links between agents. We assume also that the system is placed in an environment which the agents can access by means of their knowledge and feelings.

We distinguish four types of systems with varying degree of dynamism: static, with dynamic links, with a dynamic set of agents, and with a dynamic set of agent types (classes). Each type is more general than the preceding one. We now examine each type in more detail.

Static systems. These are usually used in approaches related to describing individual agents or stable systems. The set of agents and the links between them are fixed prior to starting the system.

Systems with dynamic links. The most common and best-studied class of systems. The set of agents is fixed before the system is started, while the links between them are established and modified during its operation. A typical example is multi-agent systems in which the interaction between agents is based on a model of negotiation in its various forms: private (two participants), tender (one consumer, many suppliers), and auction (one supplier, many consumers).

Systems with a dynamic set of agents. The process of dynamic creation and destruction of objects is both simple and complex. Its simplicity is in the ease of its description within any approach (logic, functional, or object-oriented). Its complexity is related to correctly integrating a new agent into the system, or removing an agent without compromising the integrity of the system. These problems can usually be solved in one-to-many negotiations (tenders and auctions). One can give two examples of such systems. The first example is HOMAGE [1], where an object is created by creating a new process with a special library function, and its integration into the system is described by the user through sending messages. Another example is Creché Simulation [2]. In this system, each agent is created with a certain information on its place in the community of agents; it has certain intelligence and a set of relevant knowledge. Since each agent is an element of collective intelligence, integration and removal of agents makes no problems.

We have never met *systems with a dynamically changing set of agent types*, although the situations in which an agent (an object) changes significantly its properties under certain conditions are quite common. For instance, a child grows up into an adult, and water becomes solid when it freezes. In these cases, however, one can always do with the mechanism of agent destruction/creation. On the whole, implementation of such a system is complicated, and their advantages are not obvious.

[1]This paper is supported by Russian Foundation for Basic Research (98-01-00694)

We propose a technology for development of dynamic multi-agent systems which covers the first three levels in this hierarchy and differs significantly from the above examples in the approach that it employs. Our approach, called Technology of Active Objects (or TAO, for short, see [3-5]), has some points in common with the object-oriented programming. It is based on the idea of using a single common paradigm, constraint programming, to implement intellectual, reactive, and communicative properties of agents. From the viewpoint of implementation, TAO is built as a specialized extension of the technology of subdefinite models (SD-models) developed by A.S. Narin'yani[6], which we believe to be one of the most universal, flexible, and effective approaches in constraint programming field.

1. The main properties of active objects

The notion of an "agent" is important for modern artificial intelligence [7]. Multi-agent systems developed with its help are collections of autonomous entities that possess nontrivial behavior and actively interact with each other and with the outside world. Compared to an ordinary object, an agent is more intelligent and independent. In TAO, agents are represented by the so-called active objects with the following properties:

- each active object is a record consisting of a number of slots; the set of values of the slots at a certain time is called the state of the active object;
- the behavior of an active object is described in a declarative manner, by a system of constraints; the process of solution of the system of constraints, which results in assigning some values (which can be definite or subdefinite) to all slots, is called the computation of the corresponding active object;
- an active object can react independently to the behavior of other objects, "visible" to it, and can thereby change its state;
- the behavior of an active object can depend not just on the states of other objects, but also on its own state "in the past";
- interaction between active objects is asynchronous and is based on a data-driven mechanism;
- no outside object can change the state of an active object; this can be done only by the object itself.

The set of active objects and the relation "to be outside" constitute the environment of active objects; the set of states of all objects will be called the state of the environment. The process of computation of the state of the environment of active objects is called a computation step. We assume also that there exists an abstract clock that counts the computation steps. At each step the environment goes into a new state. This means, in particular, that the state of the environment and, therefore, the states of all objects have a fixed connection with the number of the computation step. All variable slots are assumed to be unknown at the beginning of a computation step. The organization of calculations is such that the state of an active object can change at most once at each step. Each active object is recomputed at each step after the values of all of its outer objects have been calculated and cannot change their values.

To modify effectively the state of the environment, one needs additional information. Its sources in TAO are the following:

- the connection between active objects and the outside world;
- the dependence of objects on the previous state of the environment;
- modification of the environment of active objects.

Here the outside world is represented either by the user, who changes the state of the environment of active objects through input devices (keyboard, mouse, etc.), or some sensors or an arbitrary program (e.g., a DBMS), which informs the environment that new information has arrived. In all cases the

connection with the outside world is provided by special procedures written in the TAO implementation language (C++). The interface to such a procedure can be described within an active object. The active object that has a connection with the outside world is given a generic name "sensor", since it generates events for the environment of active objects. Sensors do not usually depend on other objects, and they are recomputed before other objects at each computation step.

Another important factor that influences the changes in the environment of active objects is information on its past states. One special case is the situation when the objects' behavior depends only on the previous state of the environment.

Modification of the environment of active objects means the possibility to create/destroy objects or change the relationships between them at any computation step. The dynamic component of TAO will be examined in the next section of the paper.

2. Construction of dynamic multi-agent systems

For each object, its link is represented by a set of objects that it can "see" at the moment. If this set does not change over the entire lifetime of the system, we say that the link is static, otherwise the link is dynamic. On the one hand, each agent is independent; on the other hand, it is limited by its task and its model, which defines the types of other agents that the object can "see". Real agents existing near this object are shown to it by a certain management structure containing the object as its part. This combination of agents and the facilities for controlling them is called the environment. One could say that the environment "opens agents' eyes," letting them see the outside world. It defines the separate tasks for all objects and combines them all to solve one common problem.

Below we give a fragment of the formal description of an environment.

```
Main (  Var      < Environment variables >
        Agents   < The set of agents >
        Model    < Model of the environment >
        Init     < Initial values of variables >
        Control  < Control facilities > )
```

Like an active object, the environment contains a number of variables (or slots) whose values are recomputed at each step (section **Var**). The relationships between them are described in the form of a system of constraints in the section **Model**. The initial values are given in the section **Init**. The section **Agents** lists all agents of the system. We will not describe the types of agents here; we will only say that their declarations should precede the description of the environment. Finally, the **Control** section contains the program of the system's operation. It consists of a number of instructions that should be executed asynchronously. Stripped of the syntactic detail, an instruction looks like this: $\langle condition \rangle \rightarrow \langle action \rangle$.

Here $\langle condition \rangle$ means a certain predicate on the values of environment variables and the slots of agents that have been computed in the current step, and $\langle action \rangle$ refers to a certain operation (for instance, removing an agent or setting up a link between agents). As a result of applying an instruction, a certain configuration of agents and links is prepared for the next computation step.

We describe now the algorithm of operation of the entire system. At the preparatory step, all variables and slots are given initial values, and the initial links are created. Each computation step begins with recomputing all agents in the order determined by their dependencies on each other. Next, the model of the environment is recomputed. After that, the system is rebuilt by executing all instructions. It should be noted that the links between objects can be changed only through explicit instructions. If new links are not specified for an agent, then it preserves its old links. After this section is executed, the system goes to the next computation step.

Let us now consider in detail how TAO constructs the dynamic systems of classes mentioned in the Introduction.

In *static systems*, we define a set of objects of certain types and the links between them which cannot change during operation of the system. An agent can change depending on the states of other

agents. It should be noted that TAO already has certain dynamic properties at this level provided by external stimulation through sensors.

In *limited dynamic systems* we can modify the links between agents during computations. Here all agents are known and described in advance, although they can leave temporarily the "field of vision" of the system. An example of such a system can be presented by a model of a family living in an apartment: we know all members of the family, but they are not always at home. They can enter or leave the apartment independently (they can go to work, to school, shopping, etc.). Each agent knows all others but does not always "see" them. To support implementation of such systems, the definition language for multi-agent systems contains some new facilities in addition to the mechanism of link modification: the symbol NULL for an undefined object and the actions that remove or restore an agent. The Restore action indicates the appearance of an agent which had not been used in computations for some time. The Remove action removes the agent from the environment but not from memory; the values of its slots can be saved for being restored at a later time when the agent returns to the environment.

Dynamic systems with fixed agent types can have an arbitrary number of agents of a certain type which appear and disappear in a random manner. An example of such a system is simulation of road traffic, where the number of cars and pedestrians is not known in advance. To implement such systems, one needs facilities for description of dynamic data structures and agent creation/removal. If there are many agents of one type and none of them has any advantages over the others, there is no sense in giving the agent a unique name. It is better to place it in a dynamic structure with standard access means. Such a structure in TAO is a list. To destroy an agent, we remove it from the list and free the memory occupied by the agent. To create an agent, we allocate the memory and insert a reference to it into the corresponding list. The initial values of slots of the new agent are specified if necessary.

Systems with variable types are dynamic systems in which we can modify object types, i.e. create instances of objects of types that have not been defined in advance. It is natural to ask: *Does a modified object belong to the same type that created it?* The answer is affirmative, since the basic principle of TAO is that if the type of an active object allows its dynamic modification to be made by itself, then no agent of a new type can appear in this system. The language is essentially expanded with more convenient tools for the description of the object behavior model, while the class of problems that can be solved does not grow significantly.

The above facilities are sufficient for implementation of complex dynamic multi-agent systems. In this paper we consider the case when the system consists of agents of one level. On the other hand, the entire environment can be regarded as a higher-level agent whose behavior is described by the local multi-agent system. Continuing in this manner, we can construct hierarchical multi-agent systems.

Conclusion

The technology of active objects belongs to a new generation of tools providing a comprehensive support to the development of multi-agent systems. It radically extends the capabilities of the conventional object-oriented approach and integrates it with such promising AI methods as dynamic constraint programming and distributed knowledge processing systems.

A principal novelty of TAO is that it uses constraint programming methods to specify all aspects of behavior of multi-agent systems, which allows one to simplify their design and development. The computational capabilities of TAO are based on the method of subdefinite models, which has been developed in our Institute and is presently recognized as one of the most powerful and efficient approaches in constraint programming.

The multi-agent technology that we propose can be applied to various fields, for example:

- the distributed control of technological processes;
- the support to environmental, economic, social and other types of monitoring, jointly with other components of geoinformation systems;

- the development of optimal scenarios for military and rescue operations;
- the modeling of complex processes and technical devices for design, diagnostics, and training;
- the specification of the behavior of autonomous mobile robots working in a dynamically changing environment.

References

[1] Agostino Poggi, Giovanni Adorni, *A Multi Language Environment to Develop Multi Agent Applications*, Intelligent Agents III — Proc. of the Third Intern. Workshop on Agent Theories, Architectures and Languages (ATAL-96), Springer-Verlag, Heidelberg, 1996.

[2] Darryl Davis, *Reactive and Motivational Agents: Towards a Collective Minder*, Intelligent Agents III — Proc. of the Third Intern. Workshop on Agent Theories, Architectures and Languages (ATAL-96), Springer-Verlag, Heidelberg, 1996.

[3] I. E. Shvetsov, *The Main Principles of the Technology of Active Objects*, Preprint No. 3, RRIAI, Novosibirsk, 1995, 26 p. (in Russian).

[4] I. Shvetsov, T. Nesterenko, S. Starovit, *Technology of Active Objects*, Proc. of AAAI Workshop on Constraints and Agents, Providence, USA, July 1997, 76–84.

[5] I. Shvetsov, T. Nesterenko, S. Starovit, *TAO: A Multi-Agent Technology Based on Constraint Programming*, SCAI'97, Helsinki, Finland, August 1997, 151–161.

[6] A. S. Narin'yani, *Subdefiniteness in knowledge representation and processing*, Transactions of USSR Acad. of Sciences, Technical cybernetics, Moscow, No. 5, 1986, 3–28 (in Russian).

[7] M. Wooldridge, N.R. Jennings, *Agent Theories, Architectures and Languages: a Survey*, ECAI-94, Workshop on Agent Theories, Architectures and Languages, Amsterdam, Netherlands, 1994, 1–39.