

Iliia Bider and Maxim Khomyakov

**If You Wish to Change the World, Start with Yourself.  
An Alternative Metaphor for Objects Interaction**

*Proceedings of ICEIS 2002 - the Fourth Conference on  
Enterprise Information Systems, Vol. 2, pp. 732-742,  
ICEIS Press, 2002.*

© 2002 ICEIS Press  
<http://www.iceis.org/>

# If You Wish to Change the World, Start with Yourself. An Alternative Metaphor for Objects Interaction

Ilia Bider  
IbisSoft  
Box 19567  
SE-104 32 Stockholm, Sweden  
+46 8 15 10 10  
ilia@ibissoft.se

Maxim Khomyakov  
Magnificent Seven  
1-st Baltiyskiy per. 6/21-3  
125 315 Moscow, Russia

\*Died December, 2000

## ABSTRACT

During the past ten years, requirements on functionality of business applications have been slowly changing. This shift consists of moving from traditional command based applications to inherently interactive applications of workflow and groupware type. For modeling new kind of business applications, the authors suggest an approach to defining interaction that is not based on explicit communication. In this approach, interaction is realized via active relationships that can propagate changes from one object to another. Based on this idea, which comes from the previous research work of the authors, the paper discusses the issues of introducing “harnesses” on the interactive behavior, finding the right place for the end-users in the model, and modeling distribution of tasks between different users.

## Keywords

Object-oriented modeling, business modeling, law-based programming, man-machine interaction.

## 1. Introduction

### 1.1 Motivation - paradigm shift in business application development

During the past ten years, requirements on functionality of business applications have been slowly changing. This change may be described as moving from the traditional, “human-assisting” systems, to a new generation of “human-assisted” systems.

A *human-assisting* system helps a human being only to perform certain activities, e.g. to write a letter, to print an invoice, to complete a transaction, etc. The relations between these activities, and the aim of the whole process are beyond the understanding of the system, but are a prerogative of the human participant. In a *human-assisted* system, the roles are reversed, the system has a complete picture of the process and is involved in all activities. Only when the system cannot perform some activity on its own, it will ask the human participant for assistance.

The difference between the old and new generations is essential, and it can be traced in all aspects of system development, including user-interface, see fig. 1a,b. A human-assisting system can be compared with a powerful tool set where the user should know exactly what tool to use and how to find it when it is needed. A human-assisted system functions like an assembly line conveyor bringing the user a task that he/she is to complete and a tool to complete the task with, i.e. a word processor.

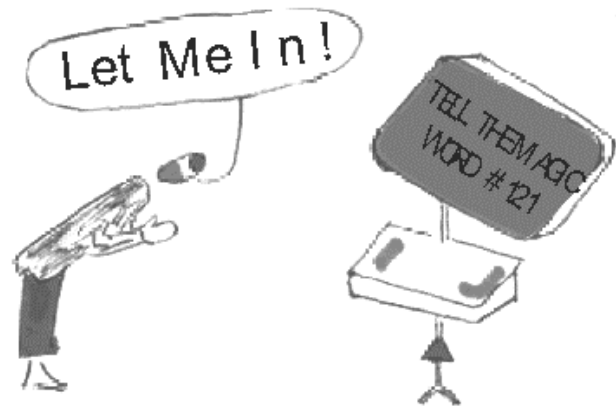


Figure 1a. Human-assisting system – a powerful toolkit

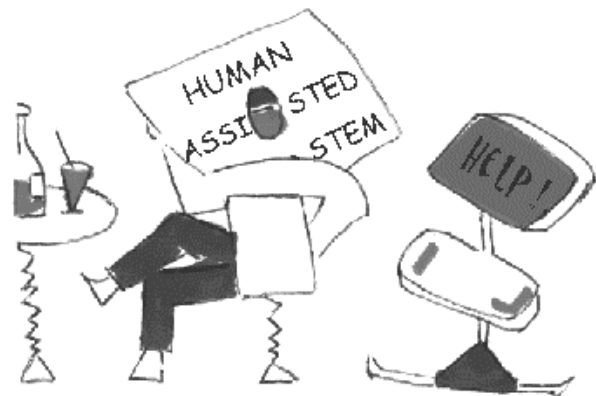


Figure 1b. Human-assisted system – an assembly Line

The difference between the generations affects also what kind of modeling techniques and languages are suitable for developing such systems. For the traditional “human-assisting system” the following is true:

1. The system is manifestly separated from the user.
2. There is only one user. Multiple users may be considered as mere repetition of the same user, with the exception of different users having different access rights. In a multi-user system, the users are not aware of each other, so the only complication for the systems development here is to solve conflicts when the users access the common database.

3. The system is purely reactive, the execution always starts on the user's command.

These properties are reflected in the basic notions of the current modeling techniques and languages, like *open system* (the system knows nothing beyond its boundaries), *use case* (somebody is using the system), *message* (a signal requesting some action), etc. The essence of such modeling is to model the software system rather than the whole business context where the system is only a part. The technical focus of modeling is reflected in how the decomposition of the system in smaller units (e.g., objects, components, etc.) is done and how their behavior is described. For example:

- The decomposition is driven by technical reasons, like geographical distribution of the system, users and information, reusability, etc.
- The resulting units do not need to reflect the entities of the business reality. And if they do, their behavior do not need to correspond to the behavior of the entities of the real world. For example, it is not uncommon that the employee object in the model has a method of calculating his/her own wages, which does not correspond to what the employee does in the business world.

The technical purposes of modeling determine also who should/could do the modeling work. The success of modeling can be ensured only if the modelers have professional knowledge of the software systems architecture.

The properties of the "human-assisted" systems are different:

1. There is no clear separation between the user and the system. Even after the system has been installed, some functions meant for the computer may be handed back to the user. And vice versa, some functions meant for the human being may be handed to the computer.
2. Multiple users is an essential property. The system should know to which user to turn when requesting assistance of a certain kind (workflow problem). Furthermore, the users might be aware of presence/absence of each other when they are engaged in collaborative work (groupware problem). Here, we have a situation of a "distributed user" rather than a situation of "distributed users" (in geographical sense). Term *distributed user* stresses that even when there is only one user, he/she is distributed between various tasks (e.g., playing a game with him/herself.) This view makes the system independent of the number of users currently working with it, as long as there is somebody from whom the system can get assistance.
3. The system is both reactive and proactive. Not only the user can request an action from the system, but also the system can anticipate and request an action from the user.

The properties listed above create a different set of requirements for modeling techniques and languages than the one that can be derived from the properties of the traditional business applications. First of all, the user becomes an integral part of the model. As the distribution of business tasks between the user and the system can change during the system's lifetime, the behavior of the user should be described in the same way as the behavior of other components of the model. Otherwise, redistribution of tasks will require rebuilding the model. Additionally, as the system can

request actions from the user, his/her representation can not differ much from the representation of other components of the system.

Making the user part of the model moves us from the *open system* concept to the *close system* concept (for the difference see [12,17]), and thus from modeling software systems to modeling business reality. Modeling the reality presumes that in the model, we reflect the entities that exist in the business world and their behavior. This is not the same as describing the behavior of the units of technical decomposition. As business reality becomes the main subject of modeling, involvement of business people in the modeling process increases. It would be improper to require from them to study contemporary theories of system architecture. We need to "liberate modeling from programming level abstractions", the topic extensively covered in [11].

The greatest difference between the old and the new generations of business applications consists in that the systems of the new generation are much more interactive than the systems of the old one. This puts the issue of modeling interactions in the center of discussion on which modeling techniques fit best the modern business application development.

## 1.2 Modeling interaction

The importance of interaction for software engineering is well understood in the computer science theory, see for example [17]. The most popular method of modeling the interactive behavior comes from the field of object-oriented programming. In the "classical" object-oriented approach, the interactive behavior is described via methods encapsulated in objects and invoked by messages (see, for example, [10]). This approach realizes the "signal-response" type of interaction. It proved to be useful for programming, specification and design of technical applications, and, moreover, for specification of traditional (i.e. reactive) business applications. However, this method is not adequate for specification of human-assisted applications. It is not meant for modeling business reality, and it does not consider the user to be an integral part of the model, i.e. on the same footing as other components.

The limitations of the classical object-oriented approach are well known in the computer science theory, see, for example, [12]. So-called generalized object models, which allow several objects to participate in the same action, were proposed in order to overcome part of those limitations, see, for example [9]. The notion of generalized object model has even been introduced in various object-oriented standards, see, for example [14]. However, as justly reflected in [12], these ideas have not found a proper place in the software engineering practice.

The word "interaction" means that the state and/or behavior of one of the interacting entities (objects, components, etc.) affects the state and/or behavior of other entities that participate in the interaction. The traditional object-oriented approach exploits explicit communication as a means of interaction (via objects sending messages to each other or themselves). This is not the only way of achieving interactive behavior. An alternative approach can be derived from the well-known saying "If you wish to change the world, start with yourself." This statement presumes that by changing its own state or behavior, the entity may cause changes in other entities. Naturally, this is impossible if this entity is totally isolated from others; the saying presumes that:

- each entity has many relationships with other entities,

- these relationships can propagate changes in one entity to the others without explicit communication (be it command-style, or negotiation-style communication).

Let us consider an example from fig. 2 where two balls are connected by a spring. Suppose the centers of these balls are fixed and cannot be moved. Assume that the balls are made from some elastic material, and each ball can change its shape and degree of elasticity. Presume that at some moment of time the first ball changed its shape as shown in fig. 3, and became hard, while the second ball remained to be elastic. Then the spring will force the second ball to change its shape, as shown in fig. 4.

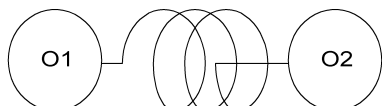


Figure 2. Two balls connected by a spring.

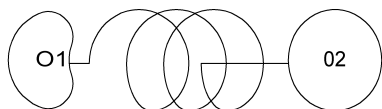


Figure 3. The first ball changed its shape and hardened.

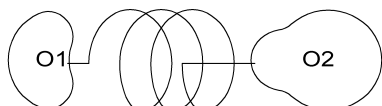


Figure 4. The spring adjusts the shape of the second ball.

The above example can be modeled as two objects connected via a computational device which we call a connector, see fig 5. The connector contains the text of a physical law that governs the spring functionality. The connector watches the objects it hangs upon, i.e., their shapes and degrees of elasticity. As soon as it discovers some changes in one or both of the objects it adjusts their shapes according to the spring law. In the real world, the spring and the balls are not floating in an empty space, but are part of some larger entity. The larger entity can also be modeled as an object in whose body the spring connector with its operands (objects the connectors hangs on) is included. The third object, *O3*, in its turn, can have active relationships with other objects. The actions of these relationships can result, for example, in the spring-balls assembly being removed from the body of *O3*.

The connector that guards some law over its operands operates on the principle of localism. Namely, it knows nothing about the objects beyond its operands, and thus, neither can change them nor use them in its work. Thus, the spring connector in fig. 5 cannot remove itself from the body of object *O3*, and fig. 5 does not cover the situation when the spring can break down as a reaction on the balls becoming too large or too small. To cover this situation, object *O3* should become an operand of the spring connector as shown in fig. 6. The model in fig. 6 allows the spring connector to remove itself from the body of the object *O3*. If at this moment objects *O1* and *O2* have no other connection to object *O3*, or its other sub-objects, they disappear from *O3* altogether.

We believe that the idea of active relationships and its realization through the concepts of law, object, and connector constitute a promising approach to modeling interactive behavior of human-assisted business applications. The natural question arises: where

is the user, is it a connector, or an object? The answer is that he/she is both.

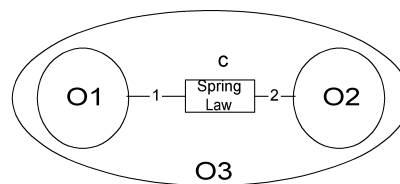


Fig. 5. The connector with its operand is part of the object *O3*.

The formal basics of the objects-connectors model hinted above are introduced in [4]. This paper is devoted to an "almost" informal discussion of how to use this model to describe interactive behavior. In several aspects the current paper goes further than [4]. This concerns the topics of distributing the users between various tasks, and making texts of laws to be an integral part of the model.

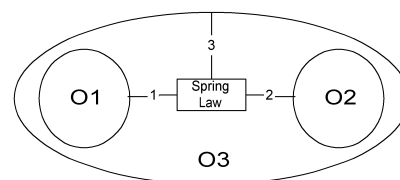


Figure 6. The model that covers the case when the spring can break down.

The rest of the paper has the following structure. In section 2, we shortly overview the basic notions of object-connectors model, and in section 3, we illustrate these notions with an example. In section 4, and 5 we discuss the place of the user in the model. In section 6, we discuss the methodological roots of our approach via comparing our metaphor with other well-known metaphors of computer science and mathematical system theory. In section 7, a list of open problems is presented that outlines the direction of our future research.

The objective of this work is to check the objects-connectors model against the requirement to cover the most essential properties of the human-assisted systems listed in section 1.1. This is done in sections 5 and 6; the results of the analysis are presented in section 7.

In the discussion of our ideas, we use a somewhat artificial example. To use the ideas in the real business practice, a two-step approach is required. First, a specialization of the model is created for a particular application domain. Then, the specialized model is used for describing business reality. This approach has been proven for the domain of business process modeling. Detail description of the specialized model for business processes is beyond the limits of this paper, see, however, [2,8].

Note that the current paper discusses only the issues of modeling the business world, and not the system architecture. It does not mean that the object-connector model cannot be used as an outline for system architecture, but we leave this topic outside the scope of the paper.

## 2. Basic notions

In this section, we give a short overview of the basic notions of the objects-connectors model. Those notions generalize the

example discussed in the previous section, and they are explained informally with the help of another example in the next section.

The objects-connectors model describes the world as consisting of:

- a set of *typed atoms*  $A$ ,
- a set of *objects*  $O$ ,
- a code of *laws*  $\Lambda$ , and
- a set of *connectors*  $CON$ , each connector hanging on a group of objects that must obey a certain law.

Atoms represent the usual notion of elementary data types, like integers, floats, strings, etc. We assume that each atom from  $A$  is assigned a type  $\tau$ . By  $A_\tau$ , we will denote the set of all atoms of the type  $\tau$ .

Atoms are the only elements of the model that can be described independently of other notions. All other notions are interconnected. The objects are used to represent complexly structured entities. At each moment of time an object can be characterized by its state. The state is not more than a set of connectors from  $CON$  included in the body of the given object.

Let  $\mathbf{R}(CON)$  be the power set of  $CON$  (a set of all subsets of connectors), and  $\perp$  a special symbol that represents the embryo state of the object. A function

$$st: O \rightarrow \mathbf{R}(CON) \hat{E} \{ \wedge \},$$

that assigns each object  $o \in O$  its state is called a *state function*. A function  $w$  that maps the time axis  $T$  into the set of all possible state functions  $ST$ :

$$w: T \rightarrow ST$$

is called a *world*. The world determines the state  $w(t)(o)$  of any object  $o$  at any point of time  $t$ . For simplicity, we consider the discrete time axis with starting point, i.e. it consists of all nonnegative integers called *time ticks*.

A *law*  $I$  is defined as a 5-tuple  $\langle name, arity, type, condition, action \rangle$ . The *name* assigns a name to the law. The *arity* is a positive integer  $n > 0$  which defines the number of objects this particular law concerns. The law can be unary, binary, etc. The type of the form  $\tau_1 \times \tau_2 \times \dots \times \tau_k$ ,  $k \geq 0$ , where  $\tau_i$  is an atomic type, defines how many atoms and of which types can be taken into consideration by the law.

Let  $I = \langle name, n, type, condition, action \rangle$  be a law. Consider a world  $w$  at a moment of time  $t$ . Let  $v = \langle v_1, \dots, v_k \rangle$  be a  $k$ -tuple of atoms of type *type*, and  $z = \langle x_1, \dots, x_n \rangle$  be an  $n$ -tuple of objects from  $O$ . Then the *condition* is a predicate that given  $v, z, w$ , and  $t$  determines whether the law with parameters  $v$  holds for objects from  $z$  in world  $w$  at time  $t$ . The *action* provides the next state of objects from  $z$  for the points for which the law is violated. In other words, the action assigns a punishment for breaking the law. The simplest type of laws is a *trivial* law for which the *condition* is always true, and the *action* is always empty.

Finally, a connector is a triple  $c = \langle I, v, z \rangle$ , where  $I$  is an  $n$ -ary law of type  $\tau_1 \times \tau_2 \times \dots \times \tau_k$ ,  $v = \langle v_1, \dots, v_k \rangle$  is a  $k$ -tuple of atoms such that  $v_i \in A_{\tau_i}$ ,  $i = 1, \dots, k$ , and  $z = \langle x_1, \dots, x_n \rangle$  is an  $n$ -tuple of objects. If the law is trivial, the connector just represents a passive relationship between its operands, eventually assigning

some properties to the relationship with the help of its  $k$  atoms (if any). This case corresponds to the usual notion of relationship from the ER model [5]. If the law is nontrivial, the connector forces the objects' structure and behavior to be in sync with each other and, possibly, with the properties of the relationship (if there are any atoms). This is the case of active relationship. If a connector's law holds on its operands, we say that the connector is *satisfied*, otherwise we say that it is *unsatisfied*.

The dynamics of the objects-connectors model can be defined by a machine in which a connector is regarded as a processing unit that monitors its operands. A connector

- *awakes* when one of its operands has been changed,
- *checks* whether the law still holds by reading the condition,
- *restores* it when it has been broken,
- *falls asleep*.

The machine has also a central unit that resolves conflicts when several connectors demand access to the same objects.

### 3. Analysis of an example

In this section, we illustrate the basic notions from the previous section by means of an example, and we discuss some important features of the model while referring to this example. The example represents a kind of distributed sort, and it is similar to the distributed exchange sort algorithm from [1] expressed in modeling language *Disco* [6].

Consider a simple promotion queue shown in fig. 7. The queue consists of a number of cells, e.g., in a hospital. The cells are ordered according to the complexity of their equipment. Each cell can have one inmate. Each inmate has a priority assigned to it. Each cell has a special measuring-device to which an inmate is connected. This device evaluates the state of the inmate and sets its priority. A new inmate can be placed in any empty cell. The queue should function in such a way that the inmates with higher priorities are placed in the better-equipped cells.

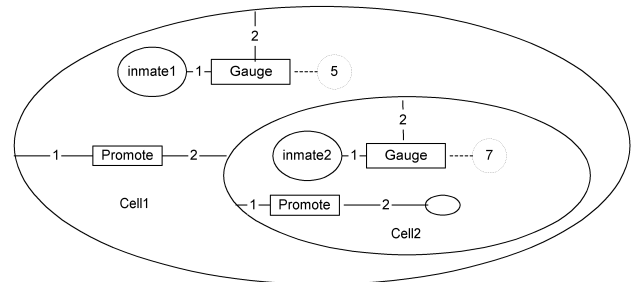


Figure 7. A promotion queue with measuring-devices.

Two laws are used for this model, *priority*, and *gauge*. The *gauge* connector is satisfied if the overall state of the *inmate* is reflected by the atomic value assigned to this connector. If the state of the inmate is not reflected by the atom, the atomic value should be changed. The legal way of changing an atom in the model is by substituting the whole connector. Thus, when the *gauge* connector is not satisfied, it replaces itself with the same gauge connector but with another atomic value assigned to it. This situation corresponds to the pointer changing its position on the dial of a real physical gauge.

The *promote* law has two operands and no atoms. It holds when a connector *priority* in its first operand assigns a lower priority to the inmate than a similar connector in its second operand. Thus, in fig. 7, the *promote* connector placed in *cell1* is satisfied. If the connector is not satisfied, it swaps inmates between its first and second operands. If the second cell is empty, it just moves the inmate from *cell1* to *cell2*. Note that if the first cell is empty, the law is always satisfied.

An example in fig. 7 illustrates how the decomposition is done in the objects-connectors model. The *gauge* connector is responsible for moving the pointer of the dial. He does it by watching the state of his first operand and evaluating it. The *promote* connector knows nothing about the internal states of the inmates included in its operands. It watches only the gauges' dials, which is enough for this connector to decide on and complete the action. In other words, the *promote* law analyzes its operands only at depth 1. It "sees" what connectors exist in its operands, including the atomic values assigned to these connectors. Actually, the idea of reduced visibility is the reason why the dial of the gauge is modeled by an atom and not by an object.

The law in our model can take into account not only the current state of its operands, but their histories too. The *promote* law does not need any historical information, its *condition* and *action* are based on the current readings of the gauges. The *gauge* itself, however, may take into account historic information. For example, if we evaluate the state of the patient in the hospital, his absolute high temperature may not be enough to assign him a high priority; we need to know whether the temperature was rising, falling, or it remained the same for some time.

The objects-connectors model is state-oriented, i.e. the state is its basic notion. The example in fig. 7 shows that state-orientation does not necessarily mean the absence of encapsulation. The idea of encapsulation is widely accepted, as it is not always possible or reasonable to know the state of an object in order to interact with it, the knowledge of the object's behavior should be enough. This idea is one of the grounds of the classical object-oriented programming. However, methods of the classical object-oriented programming are not the only approach to defining behavior. Actually, to understand the behavior of an object without looking inside it, one needs to watch some wider area in which this object operates. In our example, the *promote* connector makes its decisions based on watching the interaction between the inmate and the measuring-device inside the cell.

#### 4. Where is the user?

The model described in the previous sections looks quite deterministic. The determinism would be justified if when modeling the reality, we could have the exact knowledge of all laws that act in it. This is not true in practice where we always model only a part of the reality. There are two sources of non-determinism when modeling business reality. Firstly, the behavior of some elements of business reality cannot be fully understood or controlled because they are external for the given business, for example, customers or suppliers (see the discussion of this topic in [17]). Secondly, even the part of business reality that could be understood and controlled might be too complex to be described deterministically from the very beginning. Full understanding, if any, of this part of reality could be reached long after the business application has been designed and introduced in the operational

practice. Thus, a strategy of stepwise refinement of the model is required (for more on this topic see [12]).

To express the idea of insufficient knowledge of reality, we exploit a notion of non-deterministic law. We differentiate two types of non-determinism: firing non-determinism, and action non-determinism. For a law with *firing non-determinism*, an action can be defined even for the cases where the law holds. A connector that imposes such a law may be awoken even when none of its (visible inside the model) operands has been changed. A law with *action non-determinism* can specify two different actions for the same behavior of its operands. A connector entrusted with such a law appears to be taking different courses of action in the same situation. Of course, a law can have non-determinism of both types, firing, and action.

Connectors with non-deterministic laws, which we call boundary connectors, are our means of modeling interaction with the external world, e.g., people, or electronic devices. As part of the law is known, the external world cannot function arbitrarily. The known part of the non-deterministic laws constitutes a harness (in terminology of [17]) on the external world behavior.

Now, the user's role becomes clear, he/she is to help the connectors with non-deterministic laws. Such connector may be thought of as having a terminal where a human being can help the connector to do its job.

Let us assume that a connector's law has action non-determinism. Then it is up to the connector in case when its operands have been changed, to request the human assistance via the terminal by beeping, blinking, etc. After providing assistance, the human being may return to his other occupations, drinking coffee, for example, until the connector calls him again. Action non-determinism may vary from allowing any possible structural changes in the connector's operands to a selection from a list of predefined choices (menu).

Now, let a connector's law have firing non-determinism. Such connector can be represented as a big button that the user presses from time to time based on observations that lie beyond the boundaries of the computer system. Then it is up to the connector to do the rest of the job.

Returning to the example from the previous section, suppose the gauge from the promotion queue in fig. 7 cannot always automatically evaluate the state of the inmate, and requires human assistance from a nurse, or a doctor from time to time. To cover this case, the *gauge* law should be made non-deterministic. It might be enough to have the action non-determinism in this case. The *gauge* connector can ring the bell requesting the doctor to attend the patient and reevaluate his/her state when needed.

Another situation that can require human assistance in the promotion queue in fig.7 is when the *promote* connector cannot complete the task of switching inmates between the cells. Again, we can solve the problem by making the law of this connector non-deterministic. An alarm would ring requesting the workers to move the inmates around according to the instructions.

The above two cases illustrate that representing the user as a boundary connector insures flexibility in distribution of business tasks between the system and the user (see the properties of human-assisted systems in section 1.1). In the initial introduction of the system, some connectors may be defined as boundary, thus requiring user participation. After acquiring more knowledge on

business laws, they may be substituted by fully deterministic laws. Vice versa, some laws that seemed to be fully deterministic may prove to need human assistance after all. Switching between deterministic and non-deterministic laws will require adding/removing a user dialog script, which is a minor problem comparing to changing the model (as normally is required in the traditional approaches).

## 5. How to distribute the user?

As was explained in the previous section, the role of the user is to assist boundary connectors. This idea would be enough for modeling purposes if there were only one user in the system. He/she would be the only person to serve all connectors that require human assistance. In the normal business reality, there are many users, and they can be distributed between various business tasks in many different ways. Let us return to the example when the *gauge* connector in fig. 7 cannot automatically evaluate the state of an inmate, and requires human assistance. The distribution of human resources between different cells may vary from each cell having its own nurse (doctor) to the only person who serves all cells. The actual scheme, certainly, depends on the level of the gauge's intelligence. The more intelligent is the gauge, the less human resources it requires.

The distribution of users between various boundary connectors is not a simple task because the number of connectors and available users change in time. To cope with this task, we need to augment the objects-connectors model described in the previous sections. First of all, we need to be able to distinguish one user from another. The natural way of doing this is by representing each user as an object. Actually, the users are always part of business reality as they, normally, are employees of the company or organization whose business is to be modeled. And each employee should be represented in the model as he/she has many business processes connected to it. For example, his/her salary should be calculated monthly, his/her professional level should be maintained via various training programs, etc.

Secondly, we need a way of assigning an object that represents the user to the connector he/she is to assist. This can be done by introducing in the model a set of assignment operators  $Q$ . Different assignment operators serve to name various type of assistance the user can provide when helping a connector to do its job, e.g., a head master, assistant, etc. To each assignment operator  $q$  from  $Q$ , we attach a type in the form of  $\tau_1 \times \tau_2 \times \dots \times \tau_k$ ,  $k \geq 0$ , where  $\tau_i$  is an atomic type. The type defines the number and types of atoms that can be used to describe the assignment. The atoms can be used, for example, to define an interval of time when a particular user is to assist a particular connector. This time may be absolute, or repetitive, e.g. each day from 9 a.m. to 12 a.m.

Let  $q$  be an assignment operator of type  $\tau_1 \times \tau_2 \times \dots \times \tau_k$ ,  $k \geq 0$ , and let  $\mathbf{v} = \langle v_1, \dots, v_k \rangle$  be a  $k$ -tuple of atoms, such that  $v_i \in A_{\tau_i}$ ,  $i = 1, \dots, k$ , and  $o$  an object from  $O$ . Then the expression  $q_{\mathbf{v}}(o)$  is called a *resource allocation*. By assigning each connector that exists in the world  $w$  at time  $t$  a set of resource allocations, we complete our model with the means to express resource assignments. The assigned set can be empty, which is always the case with totally deterministic connectors. But it might be empty for boundary connectors too, which expresses the fact that human resources have not been allocated to this connector yet.

Now, we need to understand *who* will be assigning resources. This, naturally, is a connector, as it is the only element of our model that can perform actions. A new connector appears in our world only when some other connector introduces it in the course of its action. If the new connector has a non-deterministic law, human resources should be assigned to it. This may be done by the connector that introduced the given one into the world. But it might also be done by some other connector that serves as a resource manager. The latter watches the object where new boundary connectors are being introduced, and as soon as it "sees" a connector without resource allocation, it performs resource assignment.

The two situations mentioned above have analogues in the management practice. The first case corresponds to a project manager that both defines a task, and assigns an available resource to it. The second case corresponds to the matrix management, where the project manager just defines a task, and a separate resource manager finds appropriate resources to complete it. Note that both the connector that creates a given boundary connector, and the one that assigns resources (if it is separated from the first one) can be boundary connectors themselves operated by a human project (resource) manager.

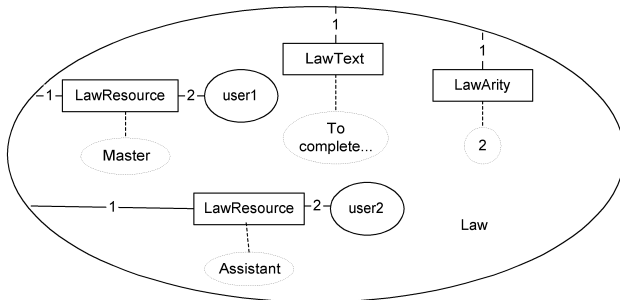
And finally, we need to understand how the assignments are done, which is the most difficult task. The simplest case of assigning resources is the assignment to "myself", which means that the creator of a new boundary connector assigns to it the same resource as it has itself.

When a task should be scheduled to somebody else, the assignment job becomes more complicated. So far, we considered any law to be independent of objects it is meant to govern. The text of the law is external to the model, and the law condition and action do not depend on particular objects, only the object's structure and behavior matter. However, when assigning resources, we need to refer to particular objects that represent the users. To cover the new challenge, additional modification of our model is required. At least some laws should become a part of the model. The natural way to do this is by representing a law as an object. An example of such representation is shown in fig. 8.

A law object in fig. 8 is built with the help of three trivial laws *LawText*, *LawAriety*, and *LawResource*. The unary connector *LawText* attaches the *text of the law* to the law object. The text of the law is expressed by an atom of some special type. This is a program written in some law programming language. The unary connector *LawAriety* defines the *arity* of the law represented by the law object. The arity is defined by an atom of type *integer*. The *LawResource* connectors attach various resources to the given law, and assign *resource names* with the help of *string* atoms. The text of the law does not refer to the actual objects. It refers only to the resource names. The *LawResource* connectors perform binding of the resource names to the actual resources that exist in the world.

To make the law object work, we need to extend the definition of connector given in section 2. Now, we allow connectors to be defined as pairs  $c = \langle o, \mathbf{z} \rangle$ , where  $o$  is an object from  $O$ , and  $\mathbf{z} = \langle x_1, \dots, x_n \rangle$  is an  $n$ -tuple of objects. We refer to this type of connectors as to *compound connectors* to differentiate them from the old-style connectors, which we call *basic connectors*.

As the law of a compound connector is an object, it can be changed by actions of other connectors. A change in the law object will force all connectors that impose this law to wake up, and check whether the changed law still holds for their operands. This check concerns both the states of the connector's operands, and the allocation of resources. Assume, for example, that the resource binding of the law in fig. 8 has been changed so that some *user3* now functions as an *Assistant* resource instead of *user2*. Then for all existing connectors that impose this law, resource assignments should be adjusted. This task is entrusted to appropriate "resource managers".



**Figure 8. A law represented as an object.**

Usually, there are several users that can do a particular type of job. When choosing one of them, the resource manager (i.e. a connector that serve as such) should take into account other assignments of this user. It means that we need to reformulate the principle of localism mentioned in section 1.2. For resource assignment, a resource manager needs to see all resource allocations made for the resources under his "supervision".

Observe that direct binding of users to a law as in fig. 8 was used only for demonstration purposes. Actual binding can be done indirectly by referring in law objects only to user roles, and separately assigning the roles to the users.

## 6. Related works

The idea of a device that guards some condition and performs some action when the condition is true is not new in the computer science theory and practice. It is expressed by such well-known metaphors as demons in AI and triggers in databases, and it is used in a number of modeling languages, see, for example, [6]. Our notion of connector realizes this idea in a slightly different fashion:

- Normally, the condition is defined by a universal predicate, which means that the guard needs to observe the whole, or a large part of the database to find any place where the condition is true. Our connector works locally, it guards only its own operands.
- The connector is the only way of expressing relationships, be it passive or active relationships. Thus, we use the same notion for describing both static data structures, and actions. The uniformity allows treating actions in the same way as static links, i.e. we can add and delete actions in the same way as we add and delete static relationships during a database transaction.
- Connectors are the only way to define actions. When a decomposition is required, the connector that guards the main law introduces connectors that guard some "local" laws

between (part of) the main connector's operands. Then it is up to these local connectors to ensure that the local laws are satisfied. In case a local law is equal to the main one, we will get the recursive decomposition. This idea was "borrowed" from the programming language Refal [16].

- We use the connector as the only way to model interaction with the external world.

There are a number of other areas where our model has common features with other approaches. For example, the idea of representing the text of a program as data has its roots in the LISP programming language. Resource allocation is one of the major themes of workflow [18]. Besides, the resource scheduling is a wide research and development area on its own, see, for example, [13]. Concerning these fields, our research is not focused on finding new approaches to solving the problems that exist in them, but rather on how to incorporate the latest results achieved in these fields into our objects-connectors model in the most natural way.

On the conceptual level, our model has a correlation with approaches widely accepted in the mathematical system theory [7] for describing physical processes. A physical process is modeled by means of differential equations. The equations define relationships between the values of state variables, i.e. a position in state space, and the values of their derivatives, i.e. the direction and speed of movement. Our definition of law represents a kind of relationships between a position in state space (limited to the contents of the connector's operands), and the direction (and partially speed) of movement in the space.

Above, we listed only the ideas, and works that had influence on our research. There is a growing bulk of literature on modeling software systems to which we have not referred. Here belong the works done in the theory of agents, architectural styles, UML, to name some of the main directions. Many features of our model can be found in these research works, or "in the air". However, the essence of our research is not the features themselves, but rather the way we are obtaining them from a small amount of simple and clear-cut notions.

It would be interesting to check each widely spread method of modeling against the properties of human-assisted systems listed in section 1.1. Such type of analysis constitutes a topic for a separate paper. In the meanwhile, we leave this exercise to the interested reader (the distributed sort example could be used for this end).

As far as we know, the only research area that directly addresses the problems of building human-assisted systems is the field of business process modeling and workflow. The difference between our approach and the traditional workflow approach to business process modeling is explained in [8].

As follows from section 1, we have doubts on applicability of current software modeling techniques to the task of modeling business reality. This opinion is based not only on the works to which we refer, but also on our long practice in business analysis and software development. In the major part of the projects, we could ourselves choose theories, methods and techniques in accordance to practical tasks we were supposed to solve. However, we also participated in projects where methods and tools were predefined by the customer. Such methods and tools included, among others, UML, and Rational Rose products. Our

experience from these projects is in the total agreement with the arguments conveyed in [11,12,17].

## 7. Concluding remarks

In section 1.1, we listed and explained three most important properties of human-assisted systems, namely:

1. no clear separation between the user and the system,
2. multiple uses distributed between various tasks,
3. proactiveness of behavior.

As was shown in section 4, the first and third properties can be expressed in our model via the notion of boundary connector. By changing the law from deterministic to non-deterministic or vice versa, we can move the responsibility for executing a task from the system to the user, or from the user to the system. Laws with action non-determinism cover the situations when the system requests help from the user (proactiveness).

As was shown in section 5, the second property requires a substantial extension of the model from [4]. Allocation operators should be introduced, and users and laws should be represented as objects. Only the last task, i.e. formal representation of laws as objects is finished at this time. Solving other tasks constitutes our research plans for the future. On the whole, the results of the analysis completed in this work make us confident that our approach is adequate for modeling human-assisted systems.

The ideas presented in this paper come from our ongoing research and practical work. Our research is focused on developing a general framework for modeling discrete dynamic systems similar to the framework that the mathematical system theory created for modeling continuous [7] and hybrid [15] dynamics. Our practical work is focused on business process modeling [8], analysis [2] and support [3].

The most common way of going from theory to practice in software engineering is: *Formal Notation* → *Case Tool* → *Code Generator*. This approach requires considerable resources in terms of manpower and money that we never had. In our practice, we used a pragmatic approach outlined below.

Our theoretical framework was used only as methodological guidelines to verify our practical decisions. We heavily relied on third party application development tools that we tuned to fit our methodology. Some parts of the software architecture were designed and implemented in a general way. For example, for storing the objects history we created an object-oriented historical database, first upon a third party record manager, then upon several commercial SQL databases. For the user interface, we created a general navigation system that allowed the user to browse through the objects' history and help the "connectors" to complete their work. This navigation system was first created for the character base terminals, and then ported to GUI. All application depended parts ("laws") were implemented in an *ad hoc* fashion (hard coding).

## 8. Acknowledgements

The work of the first author was supported by Stockholm foundation "TeknikBroStiftelsen". The preliminary version of this paper has first appeared on the website of the Journal of Conceptual Modeling ([www.inconcept.com/JCM](http://www.inconcept.com/JCM)). The authors would like to thank the anonymous reviewers for their comments on the initial draft of this work.

## 9. References

- [1] Aaltonen, T. "Example: Distributed Exchange Sort". [www.cs.tut.fi/laitos/DisCo/examples/sort/sort.html](http://www.cs.tut.fi/laitos/DisCo/examples/sort/sort.html) (1998).
- [2] Andersson, T., Andersson-Ceder, A., and Bider, I. "State Flow as a Way of Analyzing Business Processes – Case Studies", to appear in *Logistics Information Management*, Vol. 14 (1-2), MSB University Press (2002).
- [3] Bider I. "Developing Tool Support for Process Oriented Management," *Data Base Management*. 26-01-30, Auerbach, (1997).
- [4] Bider, I., Khomyakov, M. and Pushchinsky, E. "Logic of Change: Semantics of Object Systems with Active Relations", *Automated Software Engineerin.*, Vol. 7:1, pp. 9-37 (2000).
- [5] Chen P.P. "The entity-relationships model: towards a unified view of data," *ACM Trans. on Database Syst.*1(1), 9-36 (1976).
- [6] *Disco*. <http://www.cs.tut.fi/ohj/DisCo/>.
- [7] Kalman R.E., Falb P.L., and Arbib, M.A. *Topics in Mathematical System Theory*. McGraw-Hill (1969).
- [8] Khomyakov M., and Bider, I. "Achieving Workflow Flexibility through Taming the Chaos," *OOIS 2000 - 6th international conference on object oriented information systems*, Springer, 85-92 (2000). Also JCN: [www.inconcept.com/JCM/August2001/bider.html](http://www.inconcept.com/JCM/August2001/bider.html).
- [9] Kilov, H. and Ross, J. *Information Modeling: An Object-Oriented Approach*, Prentice-Hall (1994).
- [10] Korson, T., and McGregor, J.D. Understanding object-oriented: a unifying paradigm, *Communication of the ACM*, 33(9), 40-60 (September 1990).
- [11] Kurki-Suonio, R., and Mikkonen, T. "Liberating Object-Oriented Modeling From Programming-Level Abstractions," *ECOOP'97 Workshop Reader*, Springer, LNCS 1357, 195-199 (1997).
- [12] Kurki-Suonio, R. and Mikkonen, T. "Harnessing the power of interaction," *Information Modelling and Knowledge Bases*. H.Jaakkola, H. Kangassalo, E. Kawaguchi eds., IOS Press,1-11 (1999).
- [13] Podorozhny, R.M., Staudt Lerner, B., and Osterweil, L.J. "Modeling resources for activity coordination and scheduling," *Coordination Languages and Models*, Springer, LNCS 1594, 307-322 (1999).
- [14] "Reference Model for Object Data Management," *Computer Standards and Interfaces*, 15(3),124-142 (1993).
- [15] Schaft A. Van der, and Schumacher H. *An introduction to Hybrid Dynamical Systems*, Springer (2000).
- [16] Turchin, V.F. *Refal-5: Programming Guide and Reference Manual*, New England Publishing Co. Holyoke, MA. (1989).
- [17] Wegner, P. "Why interaction is more powerful than algorithms," *Communication of the ACM*, 40(5), 80-91 (May 1997).
- [18] Workflow Management Coalition. *Reference Model - The Workflow Reference Model*. WFMC-TC-1003 (19-Jan-95).