# Object Oriented Analysis and Design

**Jindřich Bůcha** [*]
email: bucha@utia.cas.cz

**Abstract:** The possibility to lay down the border between object oriented analysis and object oriented design is analyzed and the model of cooperation of these processes is presented. It is based on concepts of artificial intelligence and especially on the conceptual framework, whose original goal has been the improved efficiency of machine learning. This framework is briefly presented. It is based on an integration of all basic cognitive functions and on object oriented approaches. The model of object oriented analysis and object oriented design is built as an interpretation of this framework. This is an application of object oriented technology to artificial intelligence and vice versa.

**Keywords:** Analysis, design, object oriented, use case, artificial intelligence, machine learning

## Introduction

Recently, the discussion has taken place in object technology (OT) community.[†] It has concentrated on differences between object oriented analysis (OOA) and object oriented design (OOD), on the possibility to lay down a border between OOA and OOD. The discussion failed to make any conclusion. The **goal** of this paper is to show that this failure has been natural, that the trail to lay the border, especially to understand these processes as subsequent processes, is artificial, useless, and often difficult. It stems from current low level of understanding of OOA and OOD processes. I'll present a **model** of cooperation of these processes. It cannot be based solely on OT concepts. It's been indicated by the mentioned discussion. It is necessary to cross the OT boundaries. I'll cross them in the direction of AI (artificial intelligence). To be able to use AI, I'll have to modify it; I'll heavily use OT in AI. It results in a tight integration of OT and AI and especially machine learning concepts.

The base of this model is the framework for problem solving (PS[‡]), which is viewed from machine learning aspect mostly (Bůcha 2000a and b). Current AI systems demonstrate typically a very simple kind of learning. It can be seen from the surveys (Langley and Simon 1997; Mitchell 1998; Thrun et al. 1998), and from (Staab et al. 2000; Caravegna, Basili, and Guizauskas 2000; Boyan 1999). My approach is different. It can be characterized concisely as **"learn from learning"**. From machine learning research, we have gained insight into the suitability of using previous knowledge. It is suitable to use knowledge, which we have gathered during the exploration of learning (phenomenon), and apply it as meta-knowledge in the design of improved learning (algorithm). The body of previous knowledge is vast. What parts should be used? The simplest choice is to try to use everything. My approach is to use **all meta-knowledge that can be integrated** into an efficient learning and PS system.

To reach the specified goal I'm using OT on three levels: 1) using OT to describe PS; PS is object oriented and I use software engineering tools for its description, 2) using OT concepts

* Institute of Information Theory and Automation, Pod vodárenskou věží 4, 182 08 Prague 8, Czech Republic

† otug@Rational.Com discussion group

‡ I'll use PS for "problem solver" too.

to design PS knowledge base (KB), 3) using concept of object in its common sense (Meyer 1997) to fix fundamental assumptions about PS environment.

The body of the **paper** is organized in two main sections, the second and third. In the second section, overall PS principles are presented, PS, its knowledge base, cognitive functions, and relations among them, i.e. the object oriented PS model, are described. It is done partially by the graphical means of Unified Modeling Language (UML) (Booch, Rumbaugh, and Jacobson 1999). To model in UML, I am using a tool, Rational Rose 98i (Rational Rose 98i 1999). This section is not an easy reading. There is e.g. a lot of "*symbols*" (in *italics*) and acronyms (about 70). A more detailed description accompanied by examples can be found in (Bůcha 1999a). In the third section, I'll provide an interpretation of OOA and OOD. It can be very brief as it can rely on the model of PS. In this section, AI is applied to OT. This section can also be interpreted as an illustrative example for the previous section. In the last section, I'll provide some conclusions and an outline future work.

## Framework

In this section, I briefly describe PS framework. It is designed according to the previously identified requirements (Bůcha 1999b). It is described on the level of primary framework ***entities***. The design relates essential knowledge base (KB) structures to PS cognitive functions. An important role in this framework plays an object. Object is "everything", e.g. physical object, process, message, plan, state, goal, event, relation, and mapping.

### Principles

"A good software product is based on a small number of powerful ideas; even if it has many specialized features; they should be explainable as consequences of these basic concepts." (Meyer 1997). At the beginning, I am presenting fundamental general assumptions about the PS **context**. Just, as Meyer recommends. Brooks (1999) and Jennings (1999) do it similarly. The assumptions have the form of five **principles** and their consequences. These assumptions partially cover the requirements and form a necessary frame for them.

**The first principle**: ***Environment*** is a ***network*** of many heterogeneous, dynamic and even uncertain ***objects***.[§]

**Table 1:** ***Cognitive function goals***

| ***Cognitive function*** | ***Goal*** |
|---|---|
| ***Identification*** | *Identify model* (and possibly *problem*) using *inputs*/*outputs* from/to *Environment* |
| ***Reasoning*** | Produce *plan* to reach *PS goal* |
| ***Implementation*** | *Implement plan* by means of *inputs*/*outputs* identified/applied in *Environment* |
| ***Learning*** | Create knowledge, using *history,* to aid the solution of *PS goal* |
| ***Knowledge Base*** | Provide means to store and access *knowledge* |
| ***Self-control*** | Control *PS cognitive functions* to reach these *function goals*, to help *PS* to reach its *goal* |
| ***Initialization*** | Initiate *KB* with built-in *knowledge* |

**The second principle**: ***PS*** is an *object* of *Environment*. *PS* has a pre-specified ***goal***. If *PS* is not in *goal state*, it has ***problem***, and that should be solved by *PS*. *PS* should control

[§] Concepts like *object*, *network*, heterogeneous, dynamic, uncertain etc. are primitive concepts of the framework. I am not going to define them here; see e.g. (Meyer 1997; Bůcha 1999, 1978).

*Environment* (see Figure 1) to reach *goal*. *PS* control is accomplished by means of its ***cognitive functions*** (see Table 1). The *cognitive functions* have also their (*sub*) *goals*.

The first two *principles* have important *consequences*: The whole object oriented approach

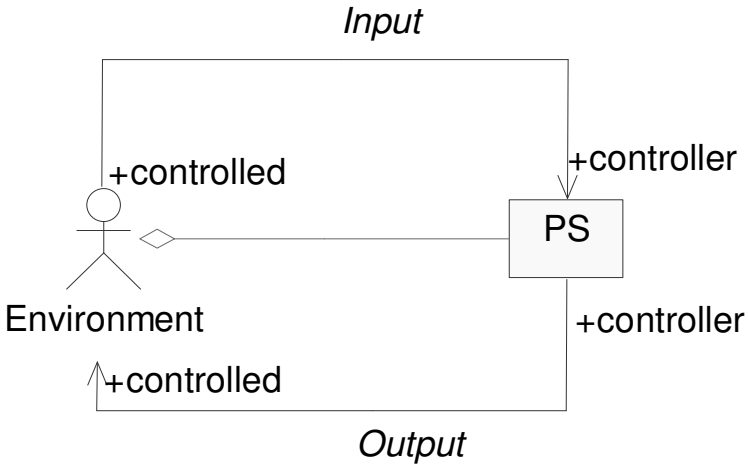


**Figure 1: Relation of PS and Environment**

(e.g. Meyer 1997) can be applied to *PS* and its parts, e.g. to *KB*. *PS* can recursively control itself (self-reflexivity). This means that *PS cognitive functions*, e.g. *reasoning*, *learning* etc. can be of a recursive kind.

**The third principle**: ***Knowledge*** is an **approximate description** (of the behavior) of *object*; *knowledge* usage increases the average probability of success of reaching *PS goal*.

These three *principles* again have *consequences*: The more accurate *goal* of *learning* is to improve a precision of *knowledge*. *Learning* is an iterative process. It should be possible to describe all kinds of *objects* of *Environment*, as the *objects* of *Environment* are interconnected and the impossibility of describing one part may affect the impossibility of describing the whole *Environment*. Thus *knowledge* should be **general** one, e.g., able to describe deterministic or stochastic, static or dynamic, simple or structured *objects*. *Learned object*, about which *PS* has already *knowledge*, can be used as *tool* for *PS*, e.g., *tool* for the next *learning* (see later). *Knowledge* can play the role of both *knowledge* and data. *Knowledge* has a role of data, when it is *object* for recursive *PS*. *PS* can have *meta-knowledge*. *Knowledge* and *PS* are more **homogenous**; i.e. *PS* does not need different kinds of *knowledge* for (external) *Environment* and for itself, e.g., for ordinary *PS objects* and *plans*.

**The fourth principle**: *Environment* is relatively very stable. During an interval of *PS* work, very little part of *Environment* is changing.

**The fifth principle**: *PS* (*learning*) should cope with its complexity.

***Knowledge base***

*PS* is a knowledge-based system. Let us start with the description of its *KB*, i.e. with the description of model of *KB entities* and semantics of these *entities*. These *entities* are e.g. *classes*, *relations* between *classes*, *class instances*, *notes*, and *class diagrams*. The UML *class diagrams*, which I am using in the paper, describe relationships among *classes*. The semantics of these *entities* are relationships between these *entities* and *objects* of *Environment*. The model of *KB* (Figure 2) consists of *Description* and other related *entities*. *Description* is *class*.

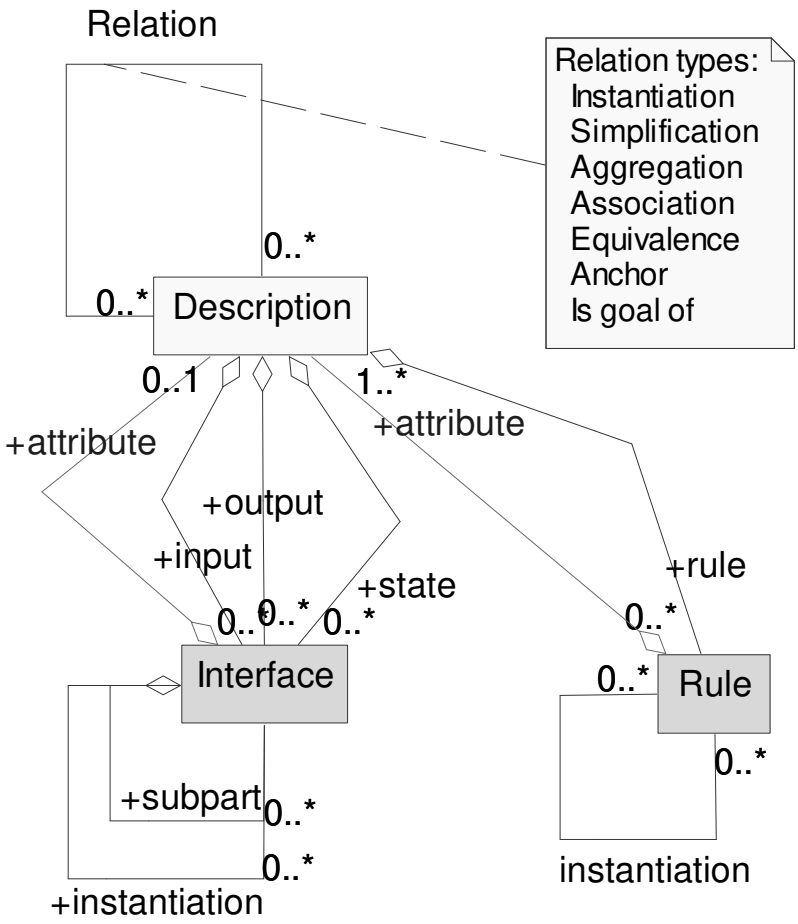


**Figure 2.** Model of knowledge base

***description***[**] is an *instantiation* of *Description*. *description* is a core of *KB*. Each *description* (indirectly) describes either *class* of *objects* (i.e. characteristics of a group of *objects*) or a partially instantiated *class* of *objects*, or a specific *object* (i.e. fully instantiated singleton *class* of *objects*). There can be a lattice of *instantiated description*s. The *instantiation* can range from not *instantiated* to fully *instantiated*. The *instantiation* is ISA (is-a kind of) *relation*. "Indirectly" means that the characteristics of *objects* can be derived via *description relations*. *Description* is *meta-class* of *class* of *objects*, i.e. *class* of *descriptions*.

By the characteristics of *objects*, I mean their relationships, interfaces, and functionality. *description* can be related to other *descriptions* via various kinds of ***relations***, e.g. *instantiation*, *simplification*, *aggregation*, *association*, *equivalence*, *anchor*, and *is-goal-of relations*. *relation* describes relationship between *descriptions*.

*description* can have *interface*(s) with +*input*, +*output*, and +*state role*s. ***interface*** is an *instantiation* of *Interface*. *interface* provides for a description of *input*, *output*, and *state* of *object*. It enables that *interface* part of *object* characteristics is partially independent of *description*. For example, a particular *interface* can be *input* for one *description* and *output* for

[**] Notice that I distinguish e.g. description in its common sense and *description* or *Description* as framework *entities*. It's important.

another one. The sharing of *interface* enables to describe a connection of two or more *objects*. Like *description*, *interface* can be *instantiated*. If there is a corresponding more general *description* (with *interface*), *interface* can be inherited. *interface* can be structured. It is accomplished by the possibility for *interface* to play the +*subpart role* for another *interface*. Each *interface* has *attribute*(s). ***attribute*** describes *input*, *output*, or *state* component of *object*. *attribute* is *description*, which plays a *role* of +*attribute* for *interface*. If there is *description* with its *attribute* and its *instantiated description* with its fully *instantiated attribute*, then this *instantiated attribute* can be ***value*** of the original *attribute*. If there is *description* with its not fully *instantiated attribute*, then this *attribute* is ***variable*** of *description*.

*description* can have also *rules*. ***rule*** is an *instantiation* of *Rule*. *rule* describes the functionality of corresponding *objects*, i.e. describes either the *state* transition or *output* production. *rule* can be generalized, i.e. related to another more general *rule* by *instantiation relation*. *rule* can be inherited if there is a corresponding more general *description* (with *rule*). *attribute* can be a part of *rule*. It is accomplished by *Description* being part of *Rule* and playing +*attribute role* in this association. I am not specifying any details of *rule* implementation. This is postponed to the later design stages. Now, you can envisage any implementation, like classical if-then rule, fuzzy rule, rule with associated probability, neural network.

***Knowledge*** consists of *description(s).*

**Problem Solver**

*PS* (Figure 3) is connected with *Environment* via its *input* and *output* (*I/O*). *I/O* are *instantiated interface*s of *PS* and *Environment*, their *attributes* have *value*s. *input values* are produced by *Environment* and a part of it is produced as *modeled inputs* (see later) by ***identification*** *function*. *PS* perceives *input*. *output values* are produced by ***implementation*** *function* to influence *Environment*. On the lowest level, *output* is a specification of a program

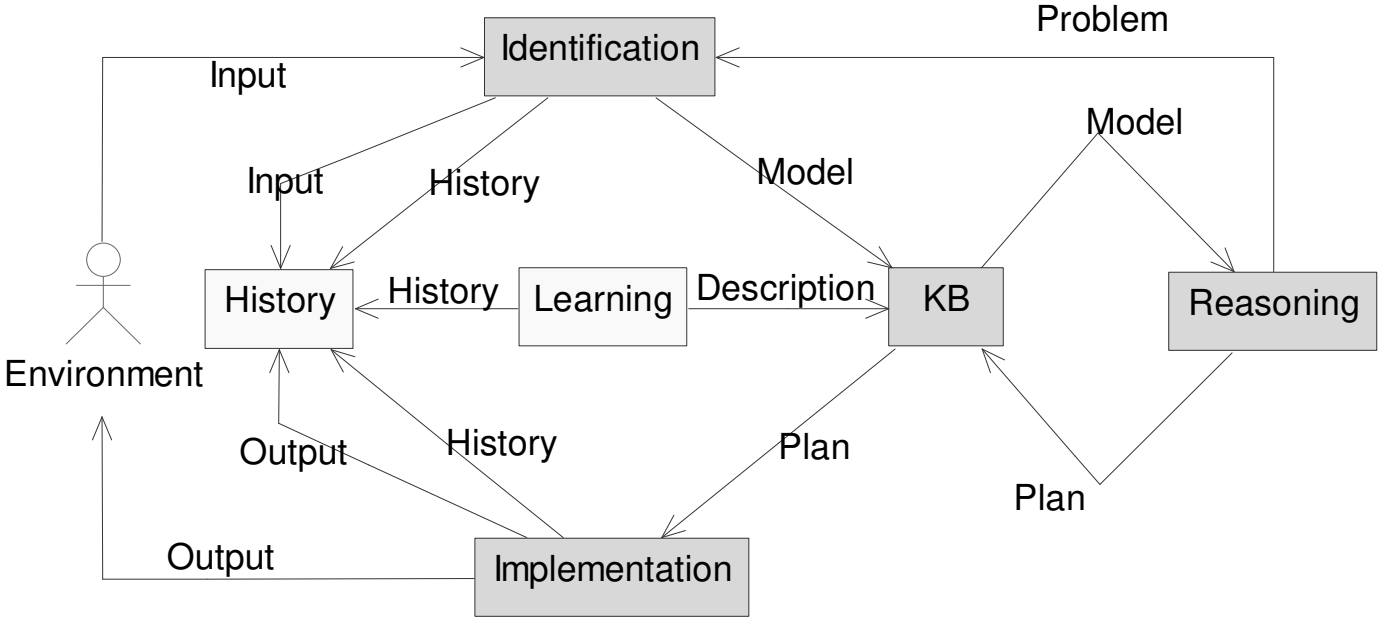


**Figure 3.** Problem solver

(or motor or actuator), that *implements* action. On higher levels, *output* is a specification of an action of some *object*. Former *values* of *I/O* (behavior of *PS* and *Environment*) are stored as ***history***. *history* is a basis for *identification*, *implementation*, and *learning functions*. *implementation* is also based on *plan* (including *plan state*).

Later, it will be seen that *implementation* produces, in some sense, an *intelligent* part of *PS output*. ***plan*** is *description*. ***reasoning*** *function* produces *plan*. It represents a possible future behavior of *PS*. *reasoning* is based on *model*. *reasoning* tries to find or construct in *KB plan* that would transmit *Environment* from current *problem* (*state*) to *goal* (*state*). It can create *sub-problems*. ***model*** is *instantiated description* of *Environment*. Parts of *I/Os* are (also) *interfaces* of *model*. *identification* tries to find *model* of *Environment*, i.e. it tries to find or construct *description* and it instantiates this *description* in such a way that it corresponds to *history*. Later, it will be seen that *identification* produces, in some sense, an *intelligent* part of *PS input*. As a part of *model*, *problem* can be identified, too. ***goal*** is a specification of set of *Environment states* that should be maintained or reached. It is described as *description*. ***learning*** produces, i.e. creates or modifies *descriptions*. *learning* is using *history* as a training set. When *learning plan*, it uses an identified *value* of *problem* for distinguishing positive and negative examples in the set.

*PS* and its *cognitive functions* are *instantiations* of *Description*. All *cognitive functions* use *KB* and are controlled by *self-control*. ***knowledge base*** *function* saves or retrieves *descriptions*. ***self-control*** *function* is to be implemented as recursive *PS*. It works like *PS*. Its environment is *Environment* together with the internals of *PS*. *I/O's* of *self-control* are *I/O's* of *PS* together with *PS cognitive functions*' *inputs*, *outputs* and *states*, e.g. allocated resources, successes and resource consumptions. The level of recursion is restricted by the extent of *KB* on the deepest level of recursion, e.g. by necessity to initialize the work of *self-control* on the next recursion level. If this *KB* extent is low, then the behavior of corresponding *cognitive function* is simple and it is not efficient to use recursive control, i.e. to initialize new recursive level. *problem* for *self-control* is an unsuccessful work of some *cognitive function*. The success can be either immediate or long-term. For example, it can be immediately revealed that *learning* cannot create *description* from a given training set. Alternatively, it can take some time to recognize that some *description* is wrong. ***initialization*** *function* initiates *KB* with built-in *knowledge*. It initializes connection between *KB* and *PS I/O attributes* and, in a similar way, it initializes connection between *KB* and *self-control I/O attributes*. It initializes the work of *self-control*, too.

*plan*, *model*, and *goal* are *instantiations* of *Plan*, *Model*, and *Goal*, which are *sub-classes* of *Description*. *history* is composed from *history* items. *history* item is a kind of *rule*, and *History* is aggregation of *sub-classes* of *Rule*.

### Shifted Border

In a simple *Environment*, to *identify* (the only one and simple) *object* and its *state* need not be a problem. To *learn description* of such *object, plan* a solution in this *Environment* described as such *object, implement plan* need not be a problem. *self-control* would not be necessary and *KB* would be trivial. I call the corresponding *cognitive functions* simple *cognitive functions*. To implement them the existing cognitive functions or ontologies from AI can be used (Thrun et al. 1998; Perez and Benjamins 1999). However, the real *Environment* is not simple, usually.

To implement (full) *cognitive functions* it is necessary to extend the simple *cognitive functions* in such a way to cover the work with object oriented *Environment*, and if necessary, to cover the work with both deterministic and stochastic *objects*, with both static and dynamic

*objects* etc. Some aspects of object oriented nature of *Environment* are solved by existing techniques, e.g. *instantiation*, *simplification*, *anchor,* and *is-goal-of relations* in *learning*. To solve other aspects, e.g. *association relations*, the concept of *tool* is used: *PS* uses a known *object* as ***tool*** that helps *PS* to perceive and/or influence *Environment*. In the range of this concept, *PS* may consider *tool* to be its part. In this way, *PS* virtually shifts its *border* with *Environment*. As it is an important aspect of the whole framework, I describe it a bit more (see Figure 4).

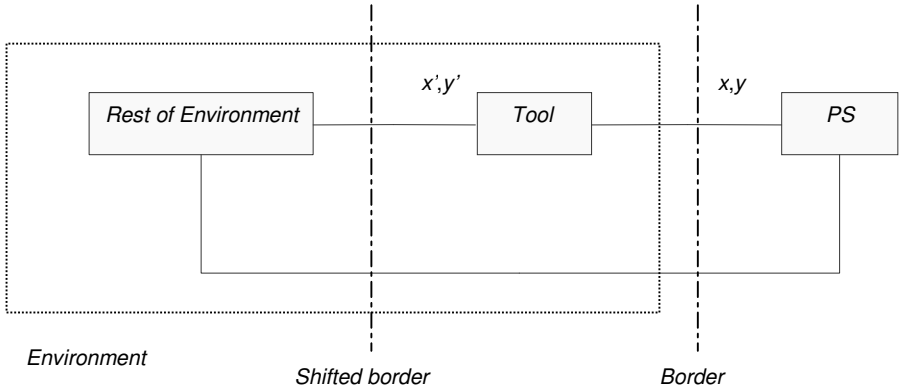


**Figure 4: Shifted border**

Let us consider *PS*, its *Environment*, some *tool*, as *object* of *Environment*, and *rest of Environment*. Let *PS* have *inputs* to perceive *Environment*; the part of them connected with *tool* is *x*. Let *PS* have *outputs* to influence *Environment*; the part of them connected with *tool* is *y*. *x* and *y* together form the part of the real *border* between *PS* and *Environment*, i.e. *border* between *PS* and *tool*. *x'* and *y'* form the part of *border* between *tool* and *rest of Environment*. *x'* and *y'* are accessible to *PS* during *learning* but not fully accessible to *PS* later. Let us assume that *PS* can *learn* characteristics of *tool* in such a way that it can later 1) *identify tool*, 2) *model* (specify) the sequence of *values* of *x'* and *y'*, if it knows the sequence of *values* of *x* and *y*. *x'* and *y'* are called *modeled inputs* and *outputs*. The equality of *modeled inputs* and *outputs*, both in *Environment* and in *PS*, can be interpreted as a shift of *border* between *Environment* and *PS*. The ***shifted border*** is partially composed from *x'* and *y'*, instead of *x* and *y*. Thus, *PS* can control *Environment* in such a way as if it has the *shifted border* with *Environment*. From the assumption that *PS* can *learn* and use tools directly connected to itself follows, that it can *learn* tools indirectly connected also. In addition, it means that *input* information for *learning*, i.e. *outputs* and *inputs*, can be iteratively extended by information about *modeled inputs* and *outputs*.

How are *PS's cognitive functions* implemented in a *shifted border* context? To ***identify*** *object* from the *rest of Environment* it is necessary to *identify tool*, *T*, and then to use *T* to get *x'*. It means to use *implementation* to apply *y* to *T*, and *simulation* to derive *x'* from *T* and *y'*, which was applied by *T*. ***simulation*** can be implemented as *self-control implementation*; its *meta-output* is not applied in the external *Environment*, but only internally, in a part of *PS model*. To ***learn*** *object* from *rest of Environment*, it is necessary to *identify tool*, *T*, to use *T* as a *tool* to get *x'* and *y'*, i.e. use *simulation* to derive *x'*, *implementation* to apply *y*, and indirectly via *tool y'*, and *learn description* on the level of *x'* and *y'*. In accordance with the *tool* concept, newly created *description* extends the set of *I/O attributes*. This, together with the use of *KB* and recursive control of *identification* and *implementation*, forms the basis of ***intelligent I/O***.

To ***implement*** *object-plan* from *rest of Environment* it is necessary to *identify tool*, *T*, use *T* to apply *y'*, i.e. use *identification* to *identify x*, *simulation* to derive *x'*, *simulation* to derive *y* from wanted *y'*, and *implementation* to apply *y* and indirectly via *tool y'*. To ***reason***, i.e. to produce *plan* for *rest of Environment* it is necessary to *identify tool*, *T*, produce *plan* on level of *x'* and *y'*; there can be some constraints implied by *T*; these can be solved either by theirs *learning* or by using the *association relation* of *descriptions*.

One of tools can be ***teacher***. One of *teachers* can be a ***society***. *PS* can use *knowledge* created by *society* of similar *PS'*s; i.e. *learning* can be of a distributed kind. Some of *tools* can be *tools* for ***communication*** with similar *PS'*s. One of *tools* can be an approach to creation or *learning*. That *tool* can be used to design *learning* itself.

In the conclusion of this section, I underline some of the most important PS characteristics: object orientation, complexity of covered environment, generality, efficiency, dynamics, uncertainty, homogeneity, integration, parallel and iterative work, intelligent I/O. They will be used (explicitly or implicitly) in the next section.

## Interpretation

In this section, I provide an interpretation of two software engineering (SE) concepts, OOA and OOD. The interpretation will be done in terms of *PS*, i.e. in terms of previous section. I'll use a mixture of *PS* and sometimes simplified SE terminology; e.g., I'll use terms man or user even if they may mean men or group of users. AS a byproduct, a tracebility of use cases will be shortly touched. It is only due to the framework of previous section that allows this section to be so short and concise.

The goal of OOA and OOD is to create some useful ***program***. Let's call this *program P*. In terms of *PS*, *P* is *object, tool*. The ***goal*** of OOA and OOD is to create useful *tool, program P*. During the gradual creation, *object P* and its *description* will go through different stages concentrated on different views, e.g. use-case, logical, implementation, process, and deployment views (Kruchten 1999).

How do we recognize that *object P* is useful? *Object P* will be created by some wo/man. Let's call her/him *PS* and let us assume that its behavior corresponds to *PS* from previous section. *PS* will have some *goal*, *G*. *Object P*, as *PS tool*, should help to reach this *goal*. This *goal* corresponds to SE's set of requirements or use cases. It corresponds to some mapping

$$g: S \rightarrow \Sigma,$$

where *S* is set of considered *states* of *Environment* and $\Sigma$ is set of subsets of *S*. Therefore, we can describe *goal G* as *object*. Recall, that *object G* can be heavily structured, as previously described. *P* will be **useful**, when it will control *Environment* to behave like *G*, i.e. when *Environment* being in *state* $s \in S$ will be transformed by *P* to *state* $s' \in g(s)$. As *description* of *G* is a *simplified description* of *Environment*, it is not specified, e.g., how long this transformation takes or how many transformation's steps it takes. Notice we have no problems with requirements like "the cost of project should be …" or "the performance of *P* should …". This is because *PS* is part of *Environment,* that when specifying requirements on *Environment,* we can specify requirements on *P* and *PS* at the same time.

For *P* being capable to control *Environment*, *PS* (the creator of *P*) should know something about *Environment,* e.g. how it can get from *state s* to *g(s)*. *PS* should know *description* of *Environment*. *P* will exist in real *Environment*, but only a part of this *Environment* will be relevant to *P*. Let us call **analysis** a process of gathering **relevant *knowledge*** about *Environment***.** As it is done in object oriented manner, as *PS* understands *Environment* in

object oriented manner, i.e. *identifies*, *reasons, learns*, uses *knowledge* of OT methods, it is **OOA**. OOA corresponds to *identification* of *model* of *Environment, M*. What is ***model*** in this case? It is *description* of *Environment*, which sufficiently precisely describes *Environment* so that this *description* e.g. covers all *state* transformations required by *G*. Recall, that *identification* corresponds to intelligent *input*, ranging from sensing elementary *attributes* through more and more complex already known *descriptions*, to *learning* new *descriptions* – dependencies.

*PS* creates *description* of *G* and *Environment* gradually according to and in cooperation with a future **user** of *P*. If it was certain that the user 1) knows *G*, 2) knows *Environment*, 3) can correctly specify them at the beginning of OOA, 4) can imagine an influence of *P* on *Environment*, 5) *PS* can directly create correct *model M* and correct *program P*, 6) *P* does not influence *E*, then OOA could have been noniterative process. None of the above listed conditions usually holds. Only gradually does the user identifies and refines her/his *goal*, only gradually does s/he *learn* about the *Environment* and *identifies* what s/he should specify. In addition, *PS* can build *P* only gradually. On top of it, *P* will be a part of *Environment*, i.e. by building *P, Environment* is changing. This change can be relevant to gathering relevant *knowledge* about *Environment*. Thus, OOA is mostly an iterative process.

Let us call **design** a process of direct building of *program P*. As it is done in object oriented manner, it is **OOD**. OOD corresponds to creation of *plan* and it is created during *reasoning*. What is ***plan*** in this case? It is *description* how to use *tool P*, *description,* which sufficiently precisely describes this *tool* so that when this *description* is *implemented* to produce a real loadable *program* and this in turn is run on computer, it carries out *state* transformations required by *G*.

The building of *program P may* depend on *model M. model M* is created iteratively. Thus, OOD may be also **iterative**. As building of *model M* may depend on *program P,* the iterativness may be inherent feature of OOA and OOD. OOA and OOD cannot be separated, with the exception of very simple projects, into separate, serially running processes. If they are separated, then some synchronization problems can occur. The nice example of *M – P* dependence is the solution of the well-known chessboard-covering problem; the chess board, without two opposite corner fields should be covered by domino pieces (Bůcha 1999).

The up-to-now presented scheme has been based on the idea of gradual building of *model M*. It was gradually extended by gradually built *plan P.* It would be possible to envisage a dual scheme. Begin from building of *program P* and gradually extend it by *descriptions* of *Environment* parts. These parts, in roles of *tools,* would help *P* to reach *G*.

The border between OOA and OOD can be sometimes arbitrary. It depends on *PS*, whether it considers some part of *Environment* to be part of *P* or not. For example, let us take *P* as an information system of some organization. The part of *P* can be employees'rules how to behave when archiving data. However, these rules can be part of *Environment* and not part of *P*. This depends also on specification of *goal G.* The complex control of *PS*, especially its recursivness, when e.g. *identification* can help to *reasoning*, and vice versa, also supports this arbitrariness.

I also interpreted the use cases (as *object G*). In this way, all characteristics of complex *object descriptions* can be applied to use cases. At the same time, the relations between *G* and *M* and *P* bring the possibility to base tracebility of use cases on these relations.

## Conclusion

In this paper, I presented the approach to OOA and OOD. It is based on both OT and AI concepts. I showed that it is possible to specify quite exactly what OOA and OOD are, however, that it is not quite suitable to try to lay down a sharp (time) border between them. As a byproduct of this endeavor, the OT/PS model of program goal, i.e. model of requirements and use cases was created. It allows specifying and understanding them in OT terms. This is important because use case is recognized as an important organizational feature of UML that has been somehow loosely attached to it, up to now.

In OOA and OOD processes, a self-reflexion plays an important role. I've not stressed it explicitly above; I'll use it here to summarize the presented approach. The approach tries to integrate two broad domains, OT and AI. It tries to invent some new understanding of (knowledge about) OOA and OOD. I can imagine that there are specialists in OT and specialists in AI with (much) deeper knowledge in these particular domains then my knowledge is. My plan for future development of the approach is to refine it with the help of such specialists. It is in accordance with one of fundamental PS principles "Knowledge is approximate" and its consequence "its precision should be improved". I also plan to persuade the OT and AI community about usefulness of presented ideas. Concerning the OT and AI field inertia, e.g. existing standards, tools, investments, it'll not be easy. However …

*This work has been partially supported by the grant GA ČR No. 102/99/1564. I also acknowledge the help of the reviewers, my daughters, and niece, who helped to improve earlier versions of the paper.*

## References

G. Booch, J. Rumbaugh, and I. Jacobson: *The Unified Modeling Language User Guide*, Reading: Addison-Wesley, 1999.

J. Boyan: IJCAI Workshop ML-1: Statistical Machine Learning for Large - Scale Optimization, Stockholm, Sweden, July 31, 1999.

R. A. Brooks, C. Breazeal (Ferrell), R. Irie, C. Kemp, M. Marjanovic, B. Scassellat and M. Williamson, Alternate Essences of Intelligence, http://www.ai.mit.edu/people/brooks/papers/group-AAAI-98.pdf. Also in AAAI-98, 1998.

J. Bůcha (a): On Efficiency of Learning: A Framework and Justification. In S. Staab et al. (eds.): ECAI Workshop Notes, Proceedings of the First Workshop on Ontology Learning (OL-2000), 14th European Conference on Artificial Intelligence, Berlin, Germany, August 22, 2000, p. 51-53, 2000.

J. Bůcha (b): On Efficiency of Learning: A Framework and Justification. Research Report No. 1983, UTIA, Prague, http://www.utia.cas.cz/ AS_dept/bucha/recent.html, 9 pp., May 2000.

J. Bůcha (a): On Efficiency of Learning, A Framework, Research Report No. 1969, UTIA, Prague, http://www.utia.cas.cz/ AS_dept/bucha/recent.html, 1999.

J. Bůcha (b): Integration of all Cognitive Functions: Requirements, http://www.utia.cas.cz/AS_dept/ bucha/ recent.html, 1999.

J. Bůcha: *Problem Solving with Knowledge Represented by Probabilistic Automata Network*, Ph.D. Thesis, Czechoslovakian Academy of Science, Prague, 110 pp., 1978 (in Czech).

F. Caravegna, R. Basili, R. Guizauskas (eds.): ECAI Workshop Notes, Workshop on Macine Learning for Information extraction, 14th European Conference on Artificial Intelligence, Berlin, Germany, August 21, 2000.

T. Dean (Ed.): *IJCAI-99, Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, Vol. 1 a 2, Stockholm, Sweden. San Francisco, Morgan Kaufmann Publishers, 1999.

N. R. Jennings: Agent-Based Computing: Promise and Perils. In (Dean 1999), p. 1429-1436, 1999.

P. Kruchten: *The Rational Unified Process, An Introduction*, Addison-Wesley, Reading, 255 p., 1999.

P. Langley, H. A. Simon: Application of Machine Learning and Rule Induction, *Communications of the ACM* **38**, no. 11, p. 55-64, November 1995.

B. Meyer: *Object-Oriented Software Construction*, Upper Saddle River, Prentice Hall, 1254 pp., 1997.

T. M. Mitchell: *Machine Learning*, McGraw Hill, N.Y., 414 pp., 1997.

J. Pearl: Reasoning with Cause and Effect. In (Dean 1999), p. 1437-1449, 1999.

A. G. Perez, V. R. Benjamins: Overview of Knowledge Sharing and Reuse Components: Ontologies and Problem-Solving Methods. In (Dean 1999), p. 1-1 – 1-15, 1999.

*Rational Rose 98i, Using Rose*, Rational Software Corporation, Cupertino, CA, 1998.

S. Staab et al. (eds.): ECAI Workshop Notes, Proceedings of the First Workshop on Ontology Learning (OL-2000), 14th European Conference on Artificial Intelligence, Berlin, Germany, August 22, 2000.

S. Thrun, C. Faloutsos, T. Mitchell, L. Wasserman: *Automated Learning and Discovery: State-Of-The-Art and Research Topics in a Rapidly Growing Field*, CMU, Pittsburgh, 1998.

M. van Someren and G. Widmer (Eds.): *Machine Learning-97, Proceedings of 9th European Conference on Machine Learning*, Prague, Czech Republic, 1997.

W. Wahlster (Ed.): *ECAI 96, 12th European Conference on Artificial Intelligence*, J. Wiley & Sons, Ltd, 1996.