KR and OOL co-operation based on semantics non reducibility

Jérôme Euzenat

INRIA Rhône-Alpes,
IMAG-LIFIA, 46, avenue Félix Viallet, 38031 Grenoble cedex 1, France
Jerome.Euzenat@imag.fr

We argue that, due to semantics non reducibility, object based-knowledge representation systems (OBKR) and object-oriented programming languages (OOL) cannot be reduced one to another. However, being aware of this incompatibility allows to organise their cohabitation and co-operation accordingly. This is illustrated through the design of a new implementation of the TROPES system.

1. WHY ARE OOL AND OBKR DIFFERENT?

There have been several works on the many possible OOL and OBKR based on the remark that OOL and OBKR are very similar at a conceptual level [16, 11, 12, 24, 4]. As a matter of fact, involved concepts include individual entities (hereby called objects), to which attributes are assigned, and generic entities (called classes) describing the structure of individual objects and organised through a relation (called specialisation). A possible source of difference, at the conceptual level, is the ability of OOL objects to do (send messages) while OBKR object only are (accessed). However, the distinction is balanced by the possibility of having access-oriented programming in OBKR, that which is computationally equivalent to message passing.

On one hand, comparing OBKR and OOL on this conceptual level, as a whole or component by component, is possible but misses the meaning of the involved concepts [16, 18]. Therefore each tentative characterisation of OOL with regard to OBKR is contradicted by some non typical system. On the other hand, in the last ten years, there has been fundamental work concerning the semantics attached to the notions of objects, classes, specialisation and so on [3, 2]. There are roughly three ways for establishing the semantics of object-based systems:

- designing an operational semantics which models the behaviour of the system without referring to the modelled domain [21];
- designing a denotational semantics given by an interpretation function ranging from the object concepts to the represented domain (it can be a one-to-one correspondence in first-order interpretations [14, 17] or an approximation through OSF-morphisms [1]). The domain can be characterised in two ways:
  - — an essentially mathematical domain (e.g. function spaces [5]);
  - — a domain which could be the reality (with nothing more than set theory; e.g. [17]); such a semantics will be called simple denotational semantics.

  The difference lays in the immediate intelligibility of the semantics by a user: what is modelled by an object is usually not a function, in consequence there is no reason why the meaning of objects should be a function.

The two first trends are mainly those used in modelling OOL — the second one being more popular in theoretical computer science — while the third one clearly concerns knowledge representation. Of course, this distinction is not that clear: some works, close to OOL, rely on a simple denotational semantics (e.g. feature algebra [22]). However, the distinction seems pretty solid: it is admitted in the OBKR field that the foundation of knowledge representation is its relationship with reality (or rather a domain to be modelled) [17], while researchers in OOL are looking for a computational model of object systems [23].

As far as the difference is made on denotational versus operational semantics, the gap is not that large. They are not contradictory, just two different ways to see the same phenomenon. But some characteristics of OOL raise problems which render difficult the task to assign them a simple denotational semantics. As a matter of fact, the emphasis put on the extendibility of OOL led to the definition of as few concepts as possible in order to enable anyone to modify the

in: Roman Cunis, Hermann Kaindl, Denis De Champaux (eds.), Proc. 11th ECAI workshop on integrating object-orientation and knowledge representation, Amsterdam (NL), no pagination, 1994

system behaviour as a whole. This has the expected advantages. However, this has two important consequences for the semantics:

- It becomes very difficult to draw the boundary between the objects which belong to the modelled domain (denoting some particular individual) and those which are part of the computational machinery of the system (this is obvious when queues, windows, programs and messages are themselves objects [10]).
- The meaning of a class as denoting, unambiguously and in itself, something (usually a set of individuals) in the modelled domain cannot be established very clearly, since the behaviour of that class is dictated by numerous changes made in the meta-class, the inheritance protocol and so on. Moreover, in many systems (e.g. [13]) this protocol is programmed in another language which must have a denotational semantics so that the OOL can have one.

Of course, these problems culminate in reflective (or self-described) systems in which well-founded sets cannot be used for defining the denotation of classes[1]. The distinction drawn above is not an arbitrary distinction between these systems: it is the core of our argumentation. In order to implement its denotational semantics, a KR system has to behave accordingly with the represented domain. If anyone can legitimately modify the behaviour of the system, this is not ensured anymore. The next step of this dialectical argument consists in noting that the modelled domain itself is capable of self-representation (and self-modification), but we have been too far… Note that nothing prevents from implementing OBKR in an OOL by implementing the appropriate semantics, however, the simple denotational semantics of the OBKR cannot be straightforwardly deduced from the operational semantics of OOL.

2. WHY BRINGING THEM TOGETHER?

The conceptual similarity is a bad reason to try to unify OBKR and OOL: it is only surface similarity without semantic grounding. Of course, there are «real similarities» which are learned directly with such a unification, but it brings too much confusion. Meanwhile, there are good reasons to put them together:

- Object-oriented programming (OOP) is so widely used that it is not possible to imagine a useful KR system unable to communicate with some OOL.
- OBKR, for their part, are a good way of representing knowledge, that which is not a strength of OOL. Thus, if an application requires knowledge representation, OBKR is a good solution.
- OOP is a powerful way of developing systems: programming an OBKR in an OOL brings portability, extendibility and integrability.

In summary, OOL and OBKR are condemned to cohabitate in real applications. These are tiny but promising arguments for putting OOL and OBKR together. They certainly justify numerous attempts to (generally) embed or implement OBKR on top of OOL by taking advantage of their extendibility [19].

There is usually, in OOL applications, some part which could be considered as representing knowledge while the other is dedicated to the functioning of the system. Our purpose is to render this distinction more explicit rather than to hide it. To that extent, we start from an analysis of the different functions of classes in OBKR and show that some can be devoted to OOL classes while other concern OBKR classes. This does not provide integration of denotational and operational semantics of such systems, but bounds the domain of each of them in such a way that they are integrated but cannot interfere. The presentation is based on the OBKR TROPES [15, 8] which introduces a conceptual difference between concepts and classes.

3. TAXONOMY VERSUS ONTOLOGY

In classical OBKR, the membership of an object to a class has important consequences. Meanwhile some of them are necessary while others are contingent (exemplified by the fact that an object can change its class). The meaning of classes is too often overloaded and this raises at least two important problems:

[1] While OBKR only have problems with the semantics of a concept which refers to itself [16, 6], OOL have problems with classes which are instances of themselves.

- It merges the ontological aspects of instances (i.e. the conditions of their being: structure, integrity, identity) and the taxonomic aspects (i.e. the conditions for seeing them as a member of a class: domain restriction, relevant fields). Merging both views together raises problems of integrity: does some object change class (breaking the integrity of the object) or is it replaced by another object with similar characteristics but belonging to another class (giving up identity of objects)? [6] Classes cannot play their role of categories in an identification system without breaking the ontological link with the objects.
- There exist several different ways of classifying objects (for instance, employees in your firm are not categorised the same way by yourself or by your account agent). It is thus important to be able to consider different concurrent taxonomies of classes organising the same set of objects.

TROPES distinguishes ontological and taxonomic aspects of classes and assigns the semantics of each aspect to distinct entities with correspondent prerogatives: concepts and classes.

3.1. Concepts

Ontological prerogatives are attached to the *concept*. They are those which warrant the integrity of an object (i.e. that it cannot be modified in a way which would lead it to cease being an instance of the concept). This also includes the management of object identity. In TROPES, objects are identified by the mean of a key: a list of field values that corresponds to only one object. The usual consequences with keys are related to integrity: an object without a key cannot exist and there cannot exist two objects with the same key (thus accidental co-reference is avoided).

These prerogatives play an important role at instanciation when the object is created and registered. They have in charge the management of existential dependencies from objects to the components without which they cannot exist. During its lifetime, an object, belonging to some concept, is ensured that none of the above postulates is violated.

3.2. Classes

As soon as the classification is seen as contingent, there can be several different classifications of the instances of the same concept: in a particular firm, the individuals are not considered the same way by the project management staff, the account office and the restaurant. Therefore the same individual is seen differently under several viewpoints and each viewpoint can have its classification of the same set of individuals.

Under a particular viewpoint, an object is attached a class and can change class as soon as it has been further identified. This organisation is very useful since one can classify an object with regard only to the relevant fields for the viewpoint. As a consequence, the set of classes to be considered is small, while considering the whole lattice of possible structures (e.g. `Vegetarian-Profit-sharing-Clerk`) would confuse the user.

A *class* is (1) a projection of the structure of the concept retaining only relevant fields and, (2) a definition of constraints that objects must satisfy in order to belong to it. It thus provide the support for classification. As opposed to instanciation, objects can be attached to a class and can be detached from it at anytime. Classes also allow the expression of hypothetical knowledge in term of default values or default inference methods. The hypothetical knowledge is relevant here since, for instance, default values are given with regard to the specificity of the object class. These values are not to be seen as the true values of fields but as hypotheses which can be made under a particular viewpoint. As a matter of fact, default values returned from different viewpoints do not have to be identical. For instance, it could be the case that research directors are usually fat while vegetarian are usually thin: what about a vegetarian director?

4. TIGHT CO-OPERATION BETWEEN OBKR AND OOL

It is noteworthy that TROPES characteristics are a combination of those of OOL and OBKR. In fact, OOL classes have concept prerogatives (e.g. instanciation, integrity, weak typing) with instances which cannot change their class and OBKR classes have class prerogatives (e.g. default inference, strong typing) with instances which can migrate. The only missing aspects of OOL is message-passing.

This distinction between concepts and classes led to the new implementation of TROPES in the context of a CLOS-like OOL (named ICOS — thus leading to the TROPICOS implementation of TROPES). In TROPICOS, the integration is such that any TROPES object can be considered as an ICOS object and any ICOS objet could be a TROPES… value. This avoids the rupture between OOL and OBKR objects. On one hand, OOL operations (e.g. integrity enforcement but also displaying, storing) can be applied to these objects without altering their representational aspects (e.g. slot values, class membership). On the other hand, pure OOL objects can freely be referred to by OBKR objects but are not interpreted as representation of an individual in the modelled domain.

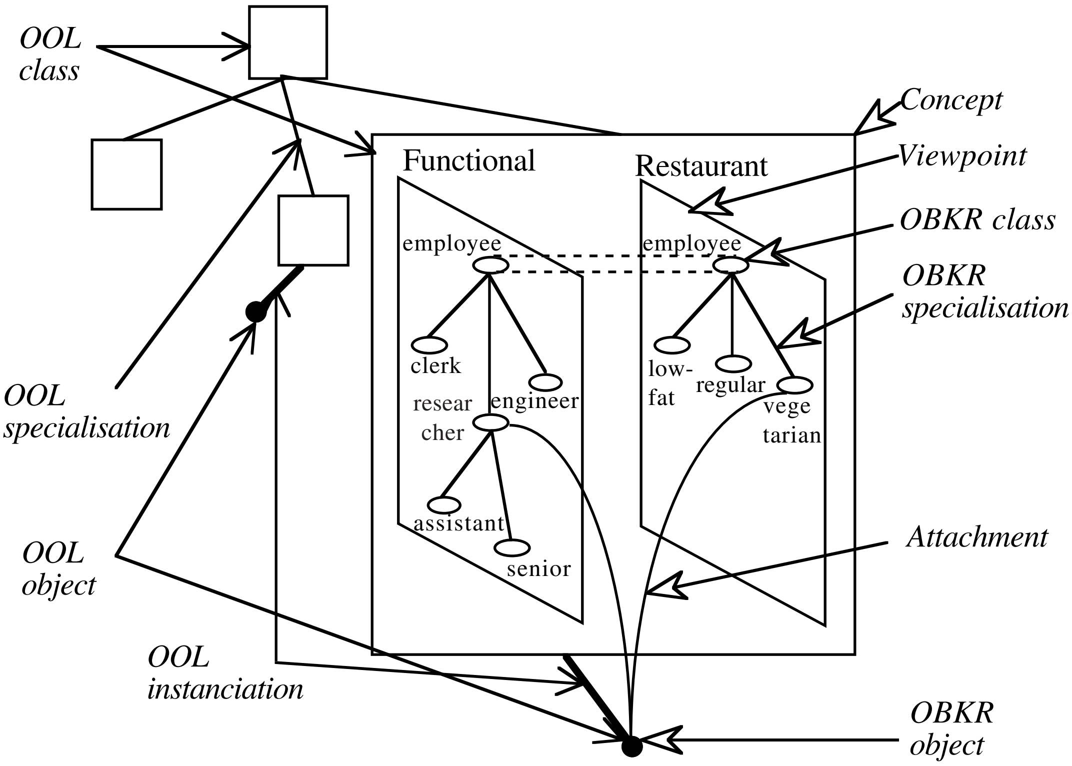

Figure. The implementation of the OBKR Tropes in an object-oriented language.

4.1.Implementing TROPES in an OOL

As a matter of fact, concepts are implemented like particular instanciable, non-mutable ICOS classes. They provide to their instances (hereby TROPES objects) the services of indexing, integrity checking (no other object can have the same key and they always have the same structure). On the other side, several viewpoints (maybe none) are attached to these concepts with a taxonomy of classes in each viewpoint. Thus, TROPES objects can be seen from several viewpoints and classified in different classes in different viewpoints. They can be reclassified at any time and their aspect (visible fields) can change from one viewpoint to another.

As a consequence of this implementation, any TROPES object is an OOL object. It behave exactly in the same way. Of course, what cannot change is the behaviour of the object from the TROPES point of view. Thus, the implementation of this object (in OOL terms) must respect TROPES semantics. However, this semantics does not extend to the whole OOL action field but only a restricted area. For instance, displaying is not taken into account by TROPES it can thus be provided by the OOL. It is expected that both sides of the object do not interact so that both semantics are preserved.

4.2.Introducing OOL objects in TROPES

So far, only TROPES objects have been fully integrated into an OOL. The converse does not hold because for OOL object to be integrated into an OBKR, they should loose their flexibility by being assigned a strict denotational semantics. However, there is a need for a continuum between OOL objects and OBKR objects which is provided by enabling the OBKR to reference

OOL objects in a minimal fashion. OOL objects are thus enabled to transit into TROPES objects (i.e. being in field values, being identified as belonging to a type) but they are not considered as TROPES objects but rather as values such as strings, boolean and so on.

For that purpose, TROPES types are declared through an abstract data type which provides a predicate for values of the type, an equality predicate and sometimes an order predicate allowing to express sub-types as enumerated domains or interval union. For instance, the `date` data type is provided with `date-p` predicate for testing if the given value is a date, a `date-equal-p` predicate for testing if two dates are equal and a `date-anterior-p` predicate. These data type are also provided with reader and writer functions. Thus, TROPES does not consider these objects as OOL objects (as a mater of fact they can be implemented as strings) but does not prohibit it. TROPES warrant that what should be a date is indeed a date and when the slot value is retrieved from a TROPES object field it can be sent the appropriate messages.

5.CONCLUSION

In summary, it has been argued that no merge is possible between OOL and OBKR but that strong pragmatic reasons appeal for co-operation. The TROPES OBKR has been briefly presented. It proposes a distinction between class and concept roles. Then, an implementation preserving differences while clarifying and strengthening links has been presented.

ROME [6] has been the first system to explicitly separate ontology from taxonomy. It makes an object instance of an "instanciation class" borrowed from OOL and let it be classified under a "representational class" related to OBKR. But, it does not separate the base into different viewpoints: there is only one taxonomy. FRAMETALK [19] proposes an implementation of frame concepts in CLOS which provides perspectives (similar to our viewpoints) [20]. Classes are just extended OOL classes and no particular semantics is provided for the system. KRS is concerned with integration of external data types in the representation systems [9]. External data are integrated through abstract data types. No distinction between OOL and KR classes is considered and no semantics is provided. Recently, it has been argued that some sort of terminological logic is more general than frames and "object-oriented data models" [4]. However, it only concerns a restricted class of object models and features has not been taken into account (e.g. reflectivity).

The presented position takes into account the current state of object semantics in OOL and OBKR. Without neglecting the efforts towards the convergence of both semantics, it represents an already practicable way of integrating both kind of systems without weakening their respective position. The main drawback of our proposal lays in the fact that an OOL implementation of OBKR is dangerous due to the flexibility of OOP.

ACKNOWLEDGEMENTS

Many thanks to Roland Ducournau for carefully reading and arguing (not that much) against my proposal. This research is partly supported by the French Groupement d'Études et Recherches sur les Génomes.

REFERENCES

[1] Hassan Aït-Kaci, Andreas Podelski, Towards a meaning of LIFE, *Journal of logic programming*16(3-4):195-234, 1993

[2] Andrew Black, Jens Palsberg, Foundations of object-oriented languages, *Sigplan notices* 29(3):3-12, 1994

[3] Ronald Brachman, What is-a is and isn't: an analysis of taxonomic link in semantic networks, *IEEE Computer* 16(10):30-36, 1983

[4] Diego Calvanese, Maurizio Lenzerini, Daniele Nardi, A unified framework for class-based representation formalisms, Proc. 4th KR, Bonn (DE), pp109-120, 1994

[5] Luca Cardelli, John Mitchell, Operations on records, *Mathematical structures in computer science* 1(1):3–48, 1991

[6] Bernard Carré and Gérard Comyn, On multiple classification, points of view and object evolution, in Jacques Demongeot, Thierry Hervé, Vincent Rialle, Christophe Roche (eds.), Artificial intelligence and cognitive sciences, Manchester University Press, Manchester (GB), pp49–62, 1988

[7] Robert Dionne, Eric Maes, Frank Oles, The equivalence of model theoretic and structural subsumption in description logics, Proc. 13th IJCAI, Chambéry (FR), pp710-716, 1993

[8] Jérôme Euzenat, A purely taxonomic and descriptive meaning for classes, Proc. IJCAI workshop «object-based representation systems» (technical report CRIN 93-R-156, Nancy (FR), 1993), Chambery (FR), pp81-92, 1993

[9] Brian Gaines, A class library implementation of a principled open architecture knowledge representation server with plug-in data types, Proc. 13th IJCAI, Chambéry (FR), pp504-509, 1993

[10] Adele Goldberg, Dave Robson, Smalltalk-80 the langage and its implementation, Addisson Wesley, Reading (MA US), 1983

[11] Hermann Kaindl, Object-oriented approach in software engineering and artificial intelligence, *Journal of object-oriented programming* 6(8):38-45, 1994

[12] Peter Karp, The design space of frame knowledge representation systems, Technical note 520, SRI AI center, Menlo Park (CA US), 1993

[13] Gregor Kiczales, Jim Des Rivières, Daniel Bobrow, The art of the meta-object protocol, The MIT press, Cambridge (MA US), 1991

[14] Hector Levesque, Ronald Brachman, Expressiveness and tractability in knowledge representation and reasoning, *Computational intelligence/intelligence informatique* 3(2):78-93, 1987

[15] Olga Mariño, François Rechenmann, Patrice Uvietta, Multiple perspectives and classification mechanism in object-oriented representation, Proc. 9th ECAI, Stockholm (SE), pp425–430, 1990

[16] Bernhard Nebel, How well a vanilla loop fit into a frame?, *Data and knowledge engineering* 1:181-194, 1985

[17] Bernhard Nebel, Reasoning and revision in hybrid representation systems, *Lecture notes in computer science (lecture notes in artificial intelligence)* 422, 1990

[18] Peter Patel-Schneider, What's inheritance got to do with knowledge representation, in Maurizio Lenzerini, Daniele Nardi, Maria Simi (eds.), Inheritance hierarchies in knowledge representation and programming languages, Wiley, Chichester (GB), pp1-11, 1991

[19] Christian Rathke, Object-oriented programming and frame-based knowledge representation, Proc. 5th. IEEE conference on tools with artificial intelligence, Boston (MA US), pp95-98, 1993

[20] Christian Rathke, David Redmiles, Multiple representation perspectives for supporting explanation in context, Research report CU-CS-645, University of Colorado, Boulder (CO US), 1993

[21] Uday Reddy, Objects as closures: abstract semantics of object-oriented programming, Proc. ACM conference on Lisp and functional programming, 1988

[22] Gert Smolka, Feature constraint logics for unification grammars, *Journal of logic programming* 12(1-2):51-87, 1992

[23] David Touretzky, The mathematics of inheritance systems, Morgan Kauffman, Los Altos (CA US), 1986

[24] Peter Wegner, Dimensions of object-based language design, Proc. 2nd OOPSLA, Orlando (FL US), pp168-182, 1987