Formalism for the Systems with Roles

Tomáš Burger, Tomáš Hruška

Brno University of Technology, Faculty of Informatics
Božetěchova 2
Brno, Czech Republic
burger@fit.vutbr.cz, hruska@fit.vutbr.cz

Abstract

In this paper we present a mathematical formalism for the so called systems with roles. We sketch the main problems and challenges for the research to be solved and we develop the generic algebraic structure, called "lattice with inheritance" and use it to describe the system with roles and to deal with the so called message dispatch problem.

Introduction and Motivation

Talking about the systems with roles, we understand the extension of traditional object-oriented systems. The main intension of this extension is to weaken the relationship of the classes and their instances (objects) and grant the objects beside the identity also real individuality. This is achieved by the possibility for the objects to be instance of multiple unrelated classes and to join or leave the classes during the lifetime of the object.

In traditional object-oriented systems (let's call them systems with classes – in contrast to the systems with roles) the object is always an instance of the single class – and the object is related to that class at the time of creation. Although this seems to be good enough for the transient objects within the standard PC applications, with rather limited lifetime and scope, this paradigm shows significant limitations, when we are talking about object-oriented architecture of the long running servers, about distributed systems with no central control, about object-oriented databases. In these cases, the expectation, that it would be enough, if the object belongs to the single class for the whole time being, is naive and this limitation significantly decreases the power of the object-oriented modeling.

Another problem related to the modeling within the systems with classes, can be called *identity versus individuality*. In object-oriented system, the object is identified by its identity, which is unique in the system and the same for the whole time being. However the fixed relationship with the single class makes the identity the only individual property of the object. The rest is completely driven by the class; the object has no control over structure of its internal status (the problem is extremely visible in the ODMG standard – see Cattel and Berry 2000, where the structure of the internal status is defined within the public ODL description). In the systems with roles, beside the identity, object holds also the strong individuality, as the structure of the internal status should be completely driven by the object and it should be only used by the different roles, the object plays.

Although there is some theoretical research mainly interested in class hierarchy modeling, where the difference between the roles and multiple inheritance is not really crucial – like in (Godin and Mili 1993 or Snasel and Benes 2002), the main concern is now about implementations of the systems with roles (Albano et al. 1993, Gottlob et al. 1996). These implementations however show lack of the theoretical backgrounds and the answers to some challenges, related to the systems with roles, are driven more by the pragmatic attitude to implement something, then by the desire to find the right answer. – That's especially true for the *message dispatch problem*, which we would like to cover in this paper.

The rest of the paper is organized as follows: section 2 describes the basic language constructs, which are necessary to transform the system with classes to the system with roles, section 3 present simple mathematical construct, called *lattice with inheritance*, section 4 uses this construct to provide formal model of the system with roles, section 5 offers a formalism for the message dispatch problem within the systems with roles, section 6 investigates the possible approaches toward the solution of this problem and finally section 7 closes the paper with some conclusions and further research plans.

Language Constructs

Looking for the programming language for the systems with roles, we would like to keep all the benefits we gain with the systems with classes. Among the key benefits belong following capabilities of the classes (for detailed reasoning about these benefits, see for instance Meyer 1997):

- To organize the related coding together
- To act as the type in the system
- To encapsulate, to hide the implementation details
- To inherit the definition from the ancestors and to form the inheritance hierarchy
- Capability of the instance of the particular class to act as instance of class' ancestors – so called polymorphism.

So, there are a lot of good reasons to keep the classes in the system. After all the roles are nothing else then classes

again, the difference is only how these classes are related to the objects in the system.

The well known operator *new* of the systems with classes does in fact two operations: it generates new object identity and it assigns this identity to the specified class. In the system with roles, these two operations are not tight together anymore and thus it is good to split them into two operators – *create* and *bind*; it is of course not forbidden to provide later on the operator *new* again as macro or sequence of the two operators in the row – especially because it is unlikely, that the objects could be treated "per se" – the only way, how to treat the object, is via the perspective of any of its roles.

The object with the initial role *Person* can be created the following way:

```
Person persJohn = create bind Person("John")
```

(It can look a bit different for every single programming language, but let's use some pseudo-language, similar somehow to the most common programming languages)

Following code adds another role – *Student* – to the person:

```
Student studJohn = persJohn bind
          Student("Technical University Brno")
```

And this one adds yet another role – *Driver*:

```
Driver drivJohn = persJohn bind
          Driver("Skoda Octavia")
```

The systems with classes with the stronger type-control offer the possibility to downcast the object reference – either driven by listed class name, like in Java (Arnold and Gosling 1996), or driven by the receiving variable, like in Eiffel (Meyer 1997). In the system with roles the casting requires greater flexibility and it can't be related anymore to the single direction – down. Thus it would be better to introduce the special operator *cast* for it to make it more explicit:

```
Driver drivJohn2 = studJohn cast Driver
```

Nowadays the de-allocation of the instance of the class is done by the garbage collector in the most common systems with classes. That can be for sure true also for the systems with roles – when there is no reference anymore to any of the roles of the object, the object – and all its roles – can be garbage-collected. However we need also special construct to drop just one role from the object. – That can't be done the same way, as there is the object, which always keeps the reference to all its roles (although it doesn't have to be the reference of the same kind as the references, stored in the variables in the programming code). So we need one more operator, which just do the work:

```
unbind drivJohn2
```

This single line of the code shows one problem: if this command is performed, the variable *drivJohn* would contain the invalid reference, because variables *drivJohn* and *drivJohn2* refer to the same role of the person John. – Perhaps for the sake of the security, this operation should not be allowed as long as there is another reference to the role *Driver* of the person John. But it can be ignored now as implementation detail.

So far about programming, in the next sections we turn our minds to the theory and mathematics.

Lattice with Inheritance

In this section, we develop generic algebraic structure, we call *lattice with inheritance*.

Within the section, $[S, \leq]$ denotes the lattice. For every $x \in S$ we define the set $x^{\uparrow} = \{y \in S \mid x \leq y\}$ – so called *main upper set* of the element x – and $x^{\downarrow} = \{y \in S \mid y \leq x\}$ – *main lower set* of the element x.

Definition. Let $\varphi \subseteq S$ is a predicate defined on S. The predicate φ is *closed for the upper sets*, if $(\forall x \in S)(x \in \varphi \rightarrow x^{\uparrow} \subseteq \varphi)$.

Definition. Let $\varphi \subseteq S$ and $\psi \subseteq S$ are predicates defined on S. We call the pair $[\varphi, \psi]$ the *upper inheritance structure*, if $\psi \subseteq \varphi$ and φ is closed for the upper sets.

Definition. Let $[\varphi, \psi]$ is an upper inheritance structure. Then for every $x \in S$ we define the set $\chi(x) = \{y \in x^{\downarrow} \mid y \in \psi\}$. The set $\chi(x)$ is called *lower override* of the element x.

Comment. Notice the reverse of the order. It would not be much interesting to investigate the elements above x, because the predicate φ is closed for the upper sets. There are probably only a few elements below x, which belongs also to ψ. – These elements are what we are looking for.

Proposition. $(\forall x \in S)(x \notin \varphi \rightarrow \chi(x) = \emptyset)$.

Proof. Be $y \in \chi(x)$. Then $y \in x^{\downarrow}$ and thus $y \leq x$ and thus $x \in y^{\uparrow}$. Also $y \in \psi$ and thus $y \in \varphi$. Because $y \in \varphi \rightarrow y^{\uparrow} \subseteq \varphi$ and $x \in y^{\uparrow}$, then $x \in \varphi$. Contradiction. ■

Definition. For every $X \subseteq S$ we define the set $\chi(X) = \cup_{x \in X} \chi(x)$; $\chi(\emptyset) = \emptyset$.

Proposition. $(\forall X \subseteq S)(\chi(X) = \{y \in \psi \mid (\exists x \in X)(y \leq x)\}$.

Proof. Let $T = \{y \in \psi \mid (\exists x \in X)(y \leq x)\}$. We prove both inclusions.

A. Firstly $\chi(X) \subseteq T$: Let $z \in \chi(X) \rightarrow (\exists x \in X)(z \in \chi(x)) \rightarrow (\exists x \in X)(z \in x^{\downarrow} \,\&\, z \in \psi) \rightarrow z \in \psi \,\&\, (\exists x \in X)(z \leq x) \rightarrow z \in T$.

B. Secondly $T \subseteq \chi(X)$: Let $z \in T \rightarrow z \in \psi \,\&\, (\exists x \in X)(z \leq x) \rightarrow (\exists x \in X)(z \in x^{\downarrow} \,\&\, z \in \psi) \rightarrow (\exists x \in x)(z \in \chi(x)) \rightarrow z \in \chi(X)$. ■

Definition. Upper inheritance structure $[\varphi, \psi]$ is *lower-bounded* if $(\forall x \in S)(\exists y \in x^{\downarrow})(x \in \varphi \rightarrow y \in \psi)$.

Proposition. Be $[\varphi, \psi]$ lower-bounded upper inheritance structure. Then $(\forall x \in S)(x \in \varphi \leftrightarrow \chi(x) \neq \emptyset)$.

Proof. A. Be $x \in \varphi$. Consider $\chi(x) = \emptyset$, it means $\{y \in x^{\downarrow} \mid y \in \psi\} = \emptyset$. So it means for every $y \in x^{\downarrow} \rightarrow y \notin \psi$, which can't be, because $[\varphi, \psi]$ is lower-bounded.

B. $\chi(x) \neq \emptyset \rightarrow x \in \varphi$. It is equivalent to $x \notin \varphi \rightarrow \chi(x) = \emptyset$, which was already proven above. ■

Comment. Everything developed so far for the upper inheritance structures and their lower overrides can be done similarly for the lower inheritance structures and their upper overrides.

Formalism of the System with Roles

We now use the structure developed in the previous section to describe the systems with roles. Instead of giving the definition at the beginning of the section, we develop the definition step-by-step. Every introduced notations and designations are valid through the whole section.

Let C, M and O be three disjoint finite sets. C is set of classes, M is set of messages and O is set of objects. Let [C, <:] be a lattice on C.

The partial order <: forming the lattice presents the relation of the subtyping and/or subclassing (assume these two relations as identical for now, as it is usual in the most of the common object-oriented systems). Perhaps the idea of the lattice doesn't fit to the imagination of the usual developer, bearing in mind the tree-structure, presenting the single inheritance. However, when we introduce the multiple inheritance and we would accept it as frequently used technique, the subclassing/subtyping relation would form the generic lattice. For relationships between the class hierarchies and lattices, see (Godin and Mili 1993).

We define two relations ~ and ≈ on the set $C \times M \cup C \times O$.

The pair $[c, m] \in \sim$ if the class c accepts the message m – it means, when the message m is part of the public interface of the class c. The pair $[c, m] \in \approx$ if the class implements the message m, so its definition contains the method for the message m.

The pair $[c, o] \in \sim$ if the object belongs to the class c. The pair $[c, o] \in \approx$ if the object has got the class/role c using the operator *bind*.

Then we define for every message $m \in M$ the lower inheritance structure $[\varphi(m), \psi(m)]$, where $\varphi(m) = \{c \in C \mid [c, m] \in \sim\}$ and $\psi(m) = \{c \in C \mid [c, m] \in \approx\}$, and for every object $o \in O$ the upper inheritance structure $[\varphi(o), \psi(o)]$, where $\varphi(o) = \{c \in C \mid [c, o] \in \sim\}$ and $\psi(o) = \{c \in C \mid [c, o] \in \approx\}$, both with the respect to the lattice [C, <:].

To keep the number of the braces in the notation lower, we use notation m-φ, m-ψ, o-φ, o-ψ and also m-χ (denoting upper override χ(m) of the lower inheritance structure [m-φ, m-ψ]) and o-χ (denoting lower override χ(o) of the upper inheritance structure [o-φ, o-ψ]).

We should verify that the required properties of the inheritance structure are satisfied within the systems with roles.

The predicate $\text{m-}\varphi \subseteq C$ denotes the fact, that the class $c \in \text{m-}\varphi$ accepts the message m. If the class $c \in C$ accepts the message, so do all the subclasses $d \in c\downarrow$, so it means that the predicate m-φ is closed for the lower sets. The predicate $\text{m-}\psi \subseteq C$ denotes the fact, that the class c has implemented method for the message m. Obviously such a class also accepts the message and thus $\text{m-}\psi \subseteq \text{m-}\varphi$, so the [m-φ, m-ψ] is lower inheritance structure, as required.

Note, that the lower inheritance structure about messages and methods shows no difference for both systems with classes and systems with roles.

The predicate $\text{o-}\varphi \subseteq C$ denotes the fact that the object o belongs to the class $c \in \text{o-}\varphi$. If the object o belongs to the class $c \in C$, then the object belongs also to all its superclasses $d \in c\uparrow$, so the predicate o-φ is closed for the upper sets. The predicate $\text{o-}\psi \subseteq C$ describes the relationship established by the operator *bind*, which binds the objects and classes together. We call these classes *the proper classes* of the object. Obviously, if the object o has the proper class c (so it means $c \in \text{o-}\psi$), the object belongs to the class c (so it means $c \in \text{o-}\varphi$) and then $\text{o-}\psi \subseteq \text{o-}\varphi$ and [o-φ, o-ψ] is the upper inheritance structure, as required.

Here the modeling already shows the difference between the systems with classes and systems with roles: o-ψ in the system with the classes contains always the single element, in the system with roles it is can be an arbitrary subclass of the class c, which fits to some particular conditions.

Especially the upper inheritance structure [o-φ, o-ψ] must be lower-bounded, because if the object belongs to the class c, then there must be the subclass d of the class c (so it means d <: c, or $d \in c\downarrow$), which is the proper class of the object.

On the other hand, the lower inheritance structure [m-φ, m-ψ] does not need to be upper-bounded. If it would be upper-bounded, than for every $c \in C$ would hold that $c \in \text{m-}\varphi \rightarrow \text{m-}\chi(c) \neq \emptyset$. But usually there are classes, for which $c \in \text{m-}\varphi$ & $\text{m-}\chi(c) = \emptyset$ for some $m \in M$ – so there are some messages, for which neither the class nor any of its superclasses can provide the implementation of the message. These classes are called *abstract classes*. The *complete classes* (the opposite term to the abstract classes) on the other hand fulfills the condition $(\forall m \in M)(c \in \text{m-}\varphi \rightarrow \text{m-}\chi(c) \neq \emptyset)$. This can be used to connect together the m-ψ and o-ψ, because the only complete classes can be used by the operator *bind*. So, it means: $(\forall c \in C)(\forall m \in M)(\forall o \in O)(c \approx o \,\&\, c \sim m \rightarrow \text{m-}\chi(c) \neq \emptyset)$, or in words: if the class c is the proper class of the object o and the class c accepts the message m, then there are some methods within the superclasses of the class c available implementing the message m. This is rather good property of the typed object-oriented languages.

Definition. The *system with roles* is the 6-tuple (C, M, O, <:, ~, ≈), where the following conditions hold:

1. The C, M, O are disjoint finite sets;
2. [C, <:] is lattice;
3. For every $m \in M$ the pair [m-φ, m-ψ] defined above is lower inheritance structure in [C, <:];
4. For every $o \in O$ the pair [o-φ, o-ψ] defined above is upper inheritance structure, moreover lower-bounded;
5. For every $c \in C$, $m \in M$, $o \in O$ the following condition holds: $c \approx o \,\&\, c \sim m \rightarrow \text{m-}\chi(c) \neq \emptyset$.

Message Dispatch Problem

Having our world in shape, we define the *message dispatch problem* now. There is a piece of the programming code (called client code). This client code holds the reference to the object $o \in O$ within the typed variable x of type of the class $c \in C$. Obviously then $c \sim o$. The client code sends the message $m \in M$ to the reference to the object $o \in O$, stored in the variable x. Consider the client code is

properly type-checked, so the condition c ~ m must hold too.

Schematically:

```
Type(c) x = o cast Type(c);
x.m();
```

We are then interested in the partial function md: $O \times M \times C \to C$, where md is defined for every triple [o, m, c] such, that c ~ o and c ~ m, and where md(o, m, c) selects the class with the best possible implementation of the message m. But what does that mean exactly?

Let d = md(o, m, c) for the rest of the section. We can express several conditions we want to be true for d:

- (Implementation) $d \approx m$
- (Semantics) $(\exists e \in C)(e \approx o \ \& \ e <: c \ \& \ e <: d)$
- (Best choice) $(\forall e \in C)(e \approx m \ \& \ e \sim o \to \neg(e <: d))$

Implementation condition is obvious: this is what we are looking for, the class knowing the implementation for the message.

For better understanding of the *Semantics* condition, let's think a while about the systems with classes and with single inheritance (like Java for instance). In such a case, there is exactly one class $e \approx o$ and the set of all its super classes $\{x \in C \mid e <: x\}$ contains c as well as d and is linearly ordered. So, it means either e <: d <: c holds or e <: c <: d holds. In the system with roles, there can be multiple classes $e \approx o$ and the sets of all their super classes form generic lattices. Then the condition about the "consistent semantic relationship" between c and d can be naturally generalized into the condition above: the object o must belong to the class d ($e \approx o \ \& \ e <: d \to d \sim o$) and the proper class e of the object o must be the subclass of both classes c and d. Then semantically the relationship between the given class c and resolved class d is somehow guaranteed.

Best choice condition is related to the polymorphism: it says that the message dispatch function should provide the implementation, which is the best possible within the all possible implementations, which are in the scope of the object. In the system with classes and with single inheritance, the set $o\text{-}\varphi = \{e \in C \mid e \sim o\}$ is linearly ordered and thus also $o\text{-}\varphi \cap m\text{-}\psi$ is linearly ordered and d is the minimal element of this final linearly ordered set (which must exists). If we loose the linear ordering in the generic case, we have to weaken the condition, so d is not the minimal element anymore, but at least, there is no smaller element with respect to the partial order <: on the class C.

Theorem. (Pessimistic Theorem on Message Dispatch) Let $d \in C$. Then d fulfills the three conditions above (Implementation, Semantics and Best Choice) if and only if $d \in \min(m\text{-}\chi(o\text{-}\chi(c)))$.

Proof. We prove both inclusions. A. Let $X = \{d \in C \mid d \approx m\}$ and $Y = \{d \in C \mid (\exists c \in C)(e \approx o \ \& \ e <: c \ \& \ e <: d)\}$. Because $o\text{-}\chi(c) = \{e \in C \mid e \approx o \ \& \ e <: c\}$, we can write $Y = \cup_{e \in o\text{-}\chi(c)}\{d \in C \mid e <: d\}$. Then $X \cap Y = \cup_{e \in o\text{-}\chi(c)}\{d \in C \mid d \approx m \ \& \ e <: d\} = \cup_{e \in o\text{-}\chi(c)} m\text{-}\chi(e) = m\text{-}\chi(o\text{-}\chi(c))$. So, if d fulfills *Implementation* and *Semantics* conditions, it belongs to the set $m\text{-}\chi(o\text{-}\chi(c))$. Assume now, that d fulfills also *Best Choice* condition, but not $d \in \min(m\text{-}\chi(o\text{-}\chi(c)))$. So, it means there must be $e \in C$, such that $e \in m\text{-}\chi(o\text{-}\chi(c))$ and e <: d. But it means there is $x \in C$ such that $e \in m\text{-}\chi(x)$, which means $e \approx m$ and x <: e, and $x \in o\text{-}\chi(c)$, which means $x \approx o$ and thus $e \sim o$. But then $e \approx m \ \& \ e \sim o \ \& \ e <: d$, which is contradiction with *Best Choice* condition.

B. Now assume $d \in \min(m\text{-}\chi(o\text{-}\chi(c)))$. Be $x \in C$ any arbitrary class such that $d \in m\text{-}\chi(x)$ and $x \in o\text{-}\chi(c)$. If $d \in m\text{-}\chi(x)$, then $d \approx m$ (*Implementation*) and x <: d. If $x \in o\text{-}\chi(c)$, then $x \approx o$ and x <: c, which all together gets *Semantics*. Now assume there is $e \in C$ such that $e \approx m \ \& \ e \sim o \ \& \ e <: d$. If e <: d and $d \in \min(m\text{-}\chi(x))$, then $e \notin m\text{-}\chi(x)$ and then either $\neg(e \approx m)$ or $\neg(x <: e)$. Because $e \approx m$ belongs among the premises, $\neg(x <: e)$ must hold. But because $x \in o\text{-}\chi(c)$ is any arbitrary such class, and because $x \approx o$ and $e \sim o$, and because $[o\text{-}\varphi, o\text{-}\psi]$ is lower-bounded, at least for one $x \in o\text{-}\chi(c)$ must hold, that x <: e. Contradiction. ■

Why the theorem is called pessimistic? Because it says, that if we take into account for the definition of the function md only the object o, the message m, it receives, and the class c, i.e. the role it plays, and we require the conditions above to hold, then the set $\min(m\text{-}\chi(o\text{-}\chi(c)))$ is the best possible result, we can get. – And we learned that for the systems with classes with single inheritance, this is enough to solve the message dispatch problem (because the set $\min(m\text{-}\chi(o\text{-}\chi(c)))$ contains only the single element) and it models well the known behavior of these systems.

If the system supports multiple inheritance the problem is usually identified already within the process of compilation, and the programmer, still present, is forced to declare the solution within the source code.

This is however not possible for the systems with roles, when this problem can occur when existing object gets another proper class – and at this time, the programmer is not available anymore. So, we desperately need the system to solve this problem on its own.

Note that the problem, which can be caused in the systems with multiple inheritance, is sub-problem of that one in the systems with roles and the solution to the later one would solve the former one automatically.

Toward the Solution of the Problem

So, what we need is to bring additional pieces of information into the message dispatch process. We are not ready to present the final solution now, but we can at least demonstrate some approaches, we would like to investigate. There are at least two candidates for additional data: context of the call and context of the bind.

Context of the Call

The client code, who sends the message to the object o, has one competence these days: it selects using the operator *cast* the class c, which is used as the starting point for the message dispatch algorithm. Attempt to take some attributes of the client and influence with them the message

dispatch algorithm itself, it is addressed in research only rarely. In (Sato and Aritsugi 2003) for instance the receiver of the message decided the message dispatch problem using the table with classification of the senders of the message. However such a solution means the great price of inflexibility. It is not acceptable to list within the every receiver of the message the taxonomy of the possible senders and build the message dispatch algorithm around it.

What we know and we can use is the following: the client code appears in certain method of certain class and the caller object belongs to this class (we don't consider the static methods for the sake of the simplicity). That can give us again the triplet [o, m, c]. Having these sender triplet and receiver triplet and knowing enough details about the class lattice, where both sender and receiver belong to, we can help ourselves to sort the elements of the set $\min(m\text{-}\chi(o\text{-}\chi(c)))$ (which depends only on the receiver) by the relevance with the sender.

Context of the Bind

Another candidate requires the extension of both the programming language, and the formalism. The idea is that the object doesn't get usually the new role without any surrounding context (in our example above the person becomes always the student of certain school), so the syntax of the operator *bind* should be extended with the context:

```
Student studJohn2 = persJohn
          bind["Technical University Brno"]
          Student()
```

That would allow the objects to have the same role multiple times for the different contexts (which would be the great extension of the modeling power of the system).

On the other hand, especially if the context would not be string (as in simplified example above), but the object or a role of the object, the relationship between the objects and their contexts can help to identify the correct method for the message. It can be either relationship between context of the receiver and context of the sender or even between the context of the receiver and the sender itself (like the object *School* – being context of the role *Student* for particular persons –sending messages to all its students).

Conclusions, Further Work

In the present paper we have introduced the systems with roles as the next generation of the object-oriented systems and their formalism. We presented the message dispatch problem within the systems with roles and offered some directions, where we are looking for the solutions.

Further work will concentrate on the following issues:

- Enhance the formalism to cover the roles with the context;
- Verify the formalism with alternative approaches and find the translation from other formalisms;
- Develop an additional structures on the lattice based on the relationship between the sender and receiver;
- Implement the experimental enhancement of the known programming language and implement the message dispatch algorithm there, illustrating our research.

Acknowledgments. The research has been supported by the Czech Ministry of Education in frame of the MSM 0021630503 Research Intention MIKROSYN: New Trends in Microelectronic Systems and Nanotechnologies, and by the Grant Agency of the Czech Republic through the grant GACR 102/05/0723: A Framework for Formal Specifications and Prototyping of Information System's Network Applications.

References

Albano, A., Bergamini, R., Ghelli, G., and Orsini, R. 1993. An Object Data Model with Roles. In *Proceedings of the 19th International Conference on Very Large Data Bases*, 39 – 51. San Francisco, CA: Morgan Kaufmann Publishers.

Snášel, V., Beneš, M. 2002. Deducing Design Class Hierarchy From Object Properties.. In *Proceedings of the 5th International Conference on Information Systems Modelling*, 203-212. Ostrava, CZ: MARQ.

Godin, R., and Mili, H. 1993. Building and Maintaining Analysis-level Class Hierarchies Using Galois Lattices. In *Proceedings of the 8th Annual Conference on Object-oriented Programming Systems, Languages, and Applications*, 394-410. New York, NY: ACM Press.

Arnold, K., and Gosling, J. 1996. The Java programming language. Reading, Mass.: Addison-Wesley.

Gottlob, G., Schrefl, M., and Röck, B. 1996. Extending Object-oriented Systems With Roles. In *ACM Transactions on Information Systems*, volume 14 , issue 3, 268-296. New York, NY: ACM Press.

Meyer, B. 1997. *Object-Oriented Software Construction*, 2nd edition. Prentice Hall.

Cattel, R.D.D., Berry, D.K. eds. 2000. *The Object Data Standard: ODMG 3.0*. San Francisco, CA: Morgan Kaufmann Publishers.

Sato, H., and Aritsugi, M. 2003. Accessee Controlled Type Selection for a Multiple-type Object. In *Proceedings of the 2003 ACM Symposium on Applied Computing*, 515-521. New York, NY: ACM Press.