Object Schizophrenia Problem in Modeling Is-role-of Inheritance

K. Chandra Sekharaiah
Distributed Object Systems Lab
Department of Computer Science & Engineering
Indian Institute of Technology
Madras India 600 036
chand@cs.iitm.ernet.in

D. Janaki Ram
Distributed Object Systems Lab
Department of Computer Science & Engineering
Indian Institute of Technology
Madras India 600 036
djram@lotus.iitm.ernet.in

ABSTRACT
Object orientation has evolved on the basis of inheritance mechanism. But, interestingly, the mechanism has assumed varied semantics over the years as many inheritance mechanisms came up in the literature. In this paper, we present the special semantics of is-role-of inheritance which is used for modeling objects that play multiple roles. Further, different symptoms of object schizophrenia problem (OSP) in modeling is-role-of inheritance are explained. We conclude that a solution to OSP in modeling the is-role-of inheritance can not avoid all the symptoms of OSP.

Keywords
Object Schizophrenia (OS), Object Schizophrenia Problem (OSP), Role Paradigm, Object Role Database System, Role Paradigm Conformance Model(RPCM)

1. INTRODUCTION
Inheritance is the distinguishing feature of object orientation. In [1, 2], a classic work on various facets of inheritance was presented by Antero Taivalsaari. In one of the earliest works on inheritance, in [3], Sakkinen dealt with resolving the problems and ambiguities of inheritance. Interestingly, he contradicted the point that inheritance is the central principle of OOP and listed several other properties of objects as more fundamental giving the highest priority to object identity. Later, subject oriented community emphasized the importance of object identity property in a similar vein for integrated, inter-operating suites of applications in [4]. Object oriented languages adopted mainly two kinds of reusing functionality- white-box reuse which is also called as class inheritance and black-box reuse which is also called as object composition. The characteristic properties of these techniques are different from each other. Because details of super class's implementation are exposed to derived classes, it is often said that "inheritance breaks encapsulation" [5]. Class inheritance is meant for capturing generalization-specialization relationship; that is, class inheritance should be used only if instances of the new class can be used in all situations where the existing class can be used. Class inheritance is supposed to capture subtyping. Subtyping is based on the substitution principle [6]. Object composition defines a new class as containing an instance of one or more existing classes. This type of class inherits nothing from the objects it contains.

Class-based and prototype-based approaches formed two contending object modeling paradigms for implementing shared behavior in object oriented systems. Object community often viewed delegation critically. Lieberman [7], who conceived delegation in exploring classless models of inheritance, viewed inheritance and delegation as two distinct mechanisms. Wegner viewed inheritance as a special case of delegation [8]. Wegner called languages that support objects as a language feature as object-based languages, those that support classless objects and delegation as delegation-based languages and such delegation-based languages that are based on prototypes as "prototypical" [8]. In [9], a hybrid model that captures both delegation and inheritance mechanisms was proposed. She refuted the claim that delegation is more powerful than inheritance. In [10], Ostermann observes that delegation is often used as a synonym for forwarding semantics and uses it differently for dynamic, object-based inheritance.

In [11], new characterization and disciplined use of delegation was proposed for split objects. Split objects are like roles. Three kinds of sharing were specified. The work in [12] took an integrative approach to class-based inheritance and inheritance by delegation. It provides object specialization wherein an object hierarchy can also be seen as a singular semantic unit. In this work, an object-role hierarchy must follow partial order. There is no fixed delegation structure as in most prototype systems. In [13], object inheritance was integrated in a class-based object oriented language, Smalltalk, for a role model. Corresponding to an abstraction hierarchy for objects with roles, object-role instance hierarchies are generated. Role-of relationship is defined in the abstraction hierarchy. However, implementation reuse is at object level in the form of object inheritance. This paper presents is-role-of inheritance as the reuse mechanism for modeling objects that play dynamically changing multiple roles. We explain various symptoms of object schizophrenia problem in modeling is-role-of inheritance.

The rest of the paper is organized as follows: The next section presents modeling the requirements for objects that play dynamically changing multiple roles. Delegation and consultation are compared and contrasted as two message forwarding mechanisms with diverse message processing semantics. The special semantics of is-role-of inheritance are presented. In section 3, the definitional details of object schizophrenia and object schizophrenia problem are presented. Section 4 explains how various symptoms of OSP are manifested in role modeling. Section 5 concludes the paper.

2. MODELING OBJECTS WITH ROLES

Roles provide dynamic composition and decomposition for object evolution. The variable part of the state and behavior of the role-playing entity is captured in terms of roles. A role is a context-dependent slice of behavior of an object. It serves as a unit of reuse and collaboration for accessing a role-playing object. Roles provide restricted and, often, complementary perspectives of a complex object. A role-playing entity can provide a role-based approach for access control. A role-based approach to object evolution can manage the object life cycle in terms of roles. Roles have been extensively used in object models [13, 12, 14]. In [15], office objects were modeled with different roles for different contexts in the lifetime of the object playing roles. In this paper, by role modeling, we mean modeling objects with roles that conform to role paradigm.

2.1 Role Paradigm: Modeling Complexity for Objects with Roles

Objects with roles have to satisfy such properties as visibility, dependency, identity, multiplicity, dynamicity, and abstractivity [16, 17, 18]. The properties were also explained in [13]. These properties have been stated time and again in literature though descriptions are different. Hence, and to avoid the paper getting too long, we do not explain the properties. This set of properties constitute what is called as role paradigm [18]. The works in [16, 13] explain how traditional object oriented techniques such as specialization, aggregation and association are not adequate to capture all the properties in the role paradigm. Role paradigm conformance is a requirement for advanced role models. A role model which conforms to the role paradigm is called *role paradigm conformance model* (RPCM). The role models in [17, 13] are RPCMs.

The role models in [14, 19] are early role models. They do not use role object identifiers and, hence, do not satisfy the multiplicity characteristic. Even some later role models [20, 11] do not satisfy the multiplicity characteristic. Consequently, they are not RPCMs. A comparison of the inadequacy of many role models regarding role paradigm conformance may be found in [21]. The works on modeling objects with roles show an evolutionary path of advancement towards RPCMs. A role-playing object that conforms to the role paradigm is called an RPCM object. To be clear, in this paper, by role modeling, we mean modeling RPCM objects.

In Figure 1, the RPCM object, John, plays Faculty role, Student role, Conference Associate subrole, Lab-in-charge subrole and Reviewer subrole. S, F, ECOOP, OOPSLA, L1, OSP_paper are role instances and subrole instances. An RPCM object has intrinsic and extrinsic properties and state. The core object, John, captures the intrinsic properties. These properties are time-invariant during the object lifetime. The role and subrole objects capture the extrinsic properties and state. They are time-variant during the lifetime of the core object. The lifeline of the core object is the lifeline of the role/subrole objects.

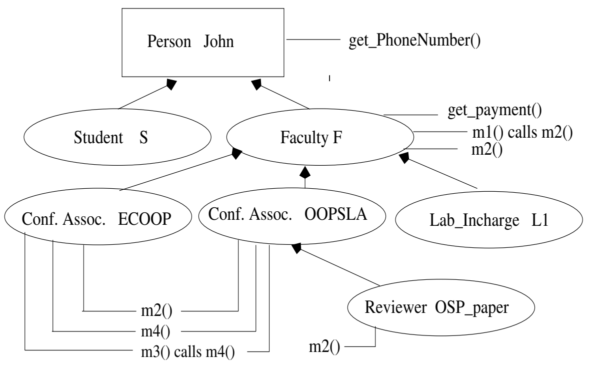

Figure 1: An Example Role Modeling Scenario: John Playing Roles

2.2 Method-lookup Mechanisms in RPCMs: Delegation Vs. Consultation

The rudimentary role models in [14, 19] do not use any message forwarding mechanism such as delegation. The later role models in [20, 11, 13, 22] use delegation as the message forwarding mechanism. In the RPCM in [17], delegation was dispensed with and messages are sent directly to the destination roles and no message forwarding mechanism is used.

A comparison of method lookup semantics in some role models is given in [21]. In the role model in [22], seven method lookup schemes (delegation paths) are used so that users can easily change to different method lookup schemes for different applications. In the role model in [20] message forwarding is based on a role and type hierarchy down and up the hierarchy by late binding. Here, the resolution mechanism for methods first goes down the hierarchy and the most specialized behaviors prevail. An object's dispatch table is updated each time a new role is acquired. In the role model in [13], when a message can not be handled by a role object, it is delegated only upward towards the more general instance in the object-role hierarchy.

2.2.1 Delegation Vs. Consultation

The message forwarding mechanisms used for sharing behavior in an object-role hierarchy are interpreted in terms of delegation or consultation. In delegation, an object, called the *master*, may have modifiable references to other objects, called its *slaves*. Messages for which the message receiver has no matching method

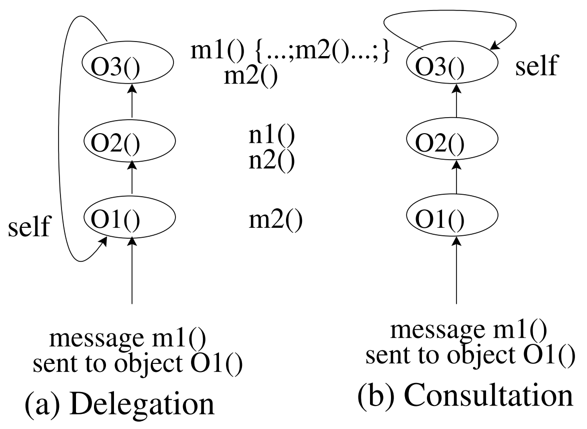

Figure 2: Delegation and Consultation: The Comparison

are automatically forwarded to its slaves. When a suitable method is found in a slave object, the method holder, it is executed after binding its implicit *self* (pseudo-variable) parameter. This parameter refers to the object on whose behalf the method is executed. Automatic forwarding with binding of *self* to the master, the message receiver, is called *delegation* whereas automatic forwarding with binding of *self* to the slave, the method holder, is called *consultation* [23].

In Figure 2, object O1 has the method m2() only, object O2 has the methods n1() and n2(), and object O3 has the methods m1() and m2() such that m1() invokes m2(). Message, m1(), sent to object, O1, is forwarded to object, O2, and then to O3. Object O1 is the message receiver and Object O3 is the method holder. We consider the *self* semantics when m1() in O3 invokes m2() during the course of its execution. In delegation-kind of message forwarding mechanism, the invocation of the method m2() in m1() of O3 will execute m2() of O1 and not m2() of O3. In consultation-kind of message forwarding mechanism, m2() of O3 becomes relevant. Figure 2 depicts the diverse *self* semantics in the two mechanisms.

2.3 Is-role-of Inheritance: Modeling Reuse for RPCMs

Is-role-of inheritance is the reuse mechanism used for RPCM objects which change their type dynamically. Smalltalk is the well-known language that offers the feature of class inheritance by delegation [1]. In [13], Smalltalk was used to implement role hierarchies for is-role-of inheritance. Is-role-of inheritance model supports type hierarchies that abstract the RPCM objects. Type hierarchies are complemented by object hierarchies. Object hierarchies are generated from the type hierarchies. In this inheritance model, specialization and inheritance are applied at the instance level. Role-of relationship exists between an object class and a role class and between a role class and other subrole classes. However, reuse is provided as object inheritance. The role hierarchies in [13] model is-role-of inheritance. A role hierarchy is different from a class hierarchy in that a subtype in a role hierarchy does not inherit the definitions of instance variables and instance methods from the supertype. Inheritance is defined at object level rather than at class level. For example, the type hierarchy in Figure 1 corresponds to Person, Student, Faculty, Conference Associate, Lab_Incharge and Reviewer. The reuse is amongst the objects shown. Is-role-of inheritance model uses a message forwarding mechanism. Either delegation or consultation can be used. In the Smalltalk implementation of Gottlob's role model, delegation-kind of message forwarding mechanism was used.

In [3], Sakkinen pointed out object identity as the most important property of an object while discussing the uniqueness principle. Is-role-of inheritance model has additional semantics as regards the identity property. In is-role-of inheritance, an object-role hierarchy represents the state and behavior of one and the same object, the RPCM object. The hierarchy satisfies the Oid integrity principle which is third dimension to Oid principle in the context of modeling is-role-of inheritance. Oid uniqueness principle and Oid persistence principle are given as two parts of the Oid principle in [24]. In object-oriented role modeling [18, 17], *Oid integrity principle* states that when an operation *find_root()* is applied to any two non-core objects in an object-role hierarchy, they should generate the same result, the core object (for instance, John) which plays roles. In Figure 1, the multiple objects, John, S, F, ECOOP, OOPSLA, LI, OSP_Paper actually form an object hierarchy that represents one and the same entity, the RPCM object, John playing roles. Oid integrity principle emphasizes the revision of the notion of object identity for RPCM objects as given in [13]. The revision brings in a dichotomy between the identity of the role-playing object and the identity of the role object. Since roles do not export behavior independent of the core object, role identity is provided by the object identity of the role-playing object.

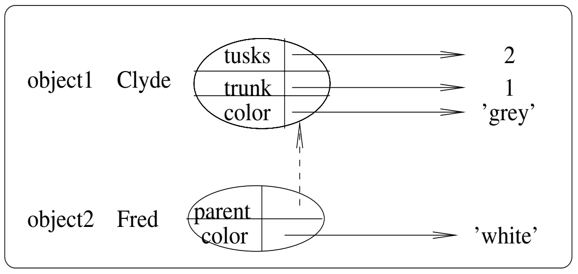

Figure 3: Prototypical Inheritance: Clyde and Fred do not Represent the Same Entity

Delegation and is-role-of inheritance are different. In delegation (prototypical inheritance), the object hierarchy need not follow Oid integrity principle. For example, the elephants Clyde and Fred of Lieberman [7] depicted in Figure 3 are two different, independent elephants though Clyde is a prototype for Fred. They have globally unique object identifiers. Fred is not existence-dependent on Clyde. Together, they do not represent one and the same entity. Clyde is accessed for the non-specialized implementation of Fred. Fred is not an extension to Clyde. The ownership of the Fred's specialization does not belong to Clyde. Consequently, the delegation hierarchy illustrated in Figure 3 does not satisfy the Oid integrity principle. On the other hand, in the is-role-of-inheritance hierarchy illustrated in Figure 1, the role and subrole objects are owned by the core object, John. Here, an object hierarchy is seen and manipulated as representing one and the same entity, John playing roles. The lower objects in the object-role inheritance hierarchy are agents for the core object. The core object and the objects in the lower levels of the hierarchy have logical, semantic identity. This is the identity characteristic specified in [16] for RPCM objects. The objects in the lower level of the hierarchy are existence-dependent on those in the higher level. This is unlike Fred's relationship with Clyde. Thus, is-role-of inheritance model satisfies the Oid integrity principle. Hence, it is different from plain delegation. Delegation and consultation are two kinds of reuse mechanisms that can be used to implement is-role-of inheritance.

The work in [19, 14] did not deal with inheritance and role identifiers in modeling objects with roles. In [24], the terms *played-by inheritance* and *role-oriented inheritance* were used for the kind of reuse mechanism required for role modeling. However, nothing similar to Oid integrity principle was specified. Further, the work is not related to modeling RPCM objects. We see is-role-of inheritance as a combination of white-box reuse and black-box reuse since a direct way of accessing the roles/subroles which are at higher level in the is-role-of inheritance hierarchy is possible as well as they can be accessed indirectly by a delegation path. Is-role-of inheritance is a tightly coupled association with extra semantics. It has two properties for reuse: transitive, asymmetric. In Figure 1, F makes black-box reuse of John, L1 makes reuse of F, and hence L1 makes black-box reuse of John. So, it is transitive. F is role of John and John is not role of F. So, it is asymmetric too. The properties hold good in general. However, is-role-of inheritance is different from is-part-of relationship for object composition. The latter is black-box reuse only.

3. OBJECT SCHIZOPHRENIA PROBLEM

Object schizophrenia problem is a problem in role model design. It is the inability of an object to communicate and interpret the messages from the client's perspective.

3.1 Object Schizophrenia

Object schizophrenia (OS) [25] is defined as the condition of an object characterized by any one or more of the features: broken interface, broken state, broken identity. By broken interface, we mean that the interface of an object under OS is an aggregate of the interfaces of multiple objects. Similar semantics apply for the terms broken state and broken identity. The prevalence of multiple states and multiple interfaces that together represent simultaneously one and the same entity indicates OS. In other words, in such a condition, the state and/or behavior of what is intended to appear as a single object are actually broken into several objects wherein each object may have its own object identity. An object in the condition of OS is called an OS object. RPCM objects described in section 2 are OS objects. For instance, in Figure 1, the role-playing object, John, is an OS object. It exports its functionality through many role, subrole objects in the form of an object-role hierarchy. OS is an inherent, common phenomenon in role modeling and some design patterns [25] such as the Adapter pattern.

3.2 OSP: What it Means

Object schizophrenia problem (OSP) is a design problem in RPCMs. An object which is under object schizophrenia and is also characterized by one or more symptoms such as broken contracts [26] (e.g. broken delegation, broken consultation), broken assumptions, dopplegangers, wrong method interpretation due to message forwarding mechanism, security problem due to message forwarding mechanism etc is said to be under OSP. RPCM design may or may not specify a message forwarding mechanism. The RPCM in [17] has not specified a message forwarding mechanism. In the presence of a message forwarding mechanism, RPCM objects are prone to OSP symptoms which are explained below.

Object schizophrenia can lead to major problems in role-oriented evolutionary software development. OS is a necessary but not sufficient condition for OSP. We relate broken semantics of abstraction, encapsulation, identity of an RPCM object with OS. Broken semantics of message passing are peculiar to OSP. OS is not a problem by itself. The usage of message forwarding mechanisms such as delegation and consultation and the associated loss of semantics of method execution in the presence of OS define OSP. Delegation and consultation entail mutually exclusive OSP symptoms in role model design. Violation of Oid integrity principle in an RPCM design leads to OSP.

The difference between intended method semantics and the processed method semantics due to OS is OSP. Wrong method semantics are not a result of delegation alone. Where the contracts in an object hierarchy are not well-defined and well-implemented, the result is OSP. OSP is the inability of an object to respond to the messages with proper information in spite of its availability. Such an inability is due to design errors in subscribing to the viewpoints in the client space.

In [25], three symptoms of OS are provided as broken delegation, broken assumptions and dopplegangers. In this paper, they are appropriately categorized as symptoms of OSP to avoid the confusion between OS and OSP. A role model has to provide non-intrusive evolution (problems can be addressed without modifying the existing code) of objects playing roles. If OSP symptoms prevail, this requirement can not be met. Hence, a role model has to be free from OSP.

4. OSP IN MODELING IS-ROLE-OF INHERITANCE

An RPCM should provide OSP-free method semantics while supporting is-role-of inheritance. In most of the role models [20, 12, 22, 13, 11], delegation is used as a common method lookup mechanism with varied semantics [21]. They are prone to object schizophrenia problem because of the usage of the message forwarding mechanism. However, they did not address OSP. Role modeling problem (RMP) is defined as the requirement to model objects with roles to conform to the role paradigm without object schizophrenia problem (OSP).

We explain five OSP symptoms below considering the role modeling scenario depicted in Figure 1. We assume delegation as the message forwarding mechanism used. However, most of the symptoms are relevant in the context of consultation too.

4.1 Wrong Message Interpretation Semantics due to Delegation

Consider that get-payment() method is there in John object as well as in Faculty role object. In role modeling based on delegation, message get-payment() sent to the Reviewer object is forwarded to the Faculty role and is executed as relevant to that role. This is wrong message interpretation semantics because the payment details for playing the Faculty role are given as the payment details for playing the Reviewer role. This kind of OSP symptom occurs in [20]. In the role model in [20], messages are sent to the most specialized behavior in the role hierarchy.

4.2 Broken Delegation

Broken delegation is a kind of misinterpretation of messages that leads to OSP. For instance, consider that an object A implements an operation by forwarding it to another object B. In case the object B makes an operation call on itself, it may find the definition of the adaptee in the class associated with B, and not in the class associated with A. In other words, there is wrong semantics if the method implementation specified in A has been overridden.

OSP symptoms could be avoided if a master identity could be established and no object identity other than the master is ever used (except within the master object). For example, to avoid the symptom of broken delegation, "self" or "this" must actually refer to the master, even in the slave objects, but most programming language implementations do not support the enforcement of such a protocol. Non-broken delegation requires a request broker as the key support by retaining and using the information needed to distinguish two Oids in the slave object usefully- *master* to refer to the identity of the master object besides the usual *this* that refers to the Oid of the *slave*.

In the aforesaid role modeling scenario, assume the following: Object John plays Faculty role object F, having methods m1() and m2(), Conference Associate subrole objects ECOOP, OOPSLA having a method m2(), and Reviewer subrole object, OSP_paper. A message m1() sent to OSP_paper will be delegated to OOPSLA and further to F. During the course of execution of m1() of F, a method m2() is invoked. Now, m1() of F will use m2() of F and not m2() of OSP_paper (which is the appropriate method for invocation). This kind of wrong method interpretation semantics is broken delegation. In this case, broken delegation degenerates to consultation. But, they are different. To illustrate, if the method holder F uses m2() of OOPSLA (its nearest object in the delegation path) in the course of execution of m1(), it is also broken delegation. Thus, broken delegation has broader semantics than consultation.

4.3 Security Problem due to Delegation

Without addressing the security concerns, delegation is used in role modeling. Where multiple delegation paths are there to the core object, different security concerns have to be figured out. Different message passing paths may require different security concerns. Where it may be secure for an object to respond to message forwarded in one delegation path, it may be insecure to respond to message forwarded in a different delegation path. In other words, security concerns of a role player object are not the same in all delegation paths. For instance, the security concerns in the message passing path L1-F-John may be different from those in the message passing path OOPSLA-F-John. A message to John via OOPSLA may not have to be allowed whereas one via L1 may be allowed.

Besides normal security provision in the object-role hierarchy, extra security concerns often arise for objects on account of delegation. For example, in Figure 1, person object, John has personal phone number. A message may be sent to OSP_paper to find out the phone number. It may be secure if clients are allowed to access the object hierarchy by delegation traversing the delegation path consisting of OSP_paper-OOPSLA-F. However, it may be insecure to allow the delegation further to John. Additional security concerns may have to be satisfied to forward the message further from F to John. However, such extra security concerns in the object hierarchy that arise due to delegation are unaddressed and unprovided by the researchers who worked on the notion of delegation. In the role modeling scenario considered, the message may be delegated unrestrictedly to the object John and the personal phone number of John is obtained. Insecure message interpretation due to the usage of message forwarding mechanism in role modeling is an OSP symptom.

4.4 Broken Assumptions

In modeling objects with roles, the symptom of broken assumptions may arise as a result of the usage of delegation as a message forwarding mechanism. The details of the symptom [25] are given below and illustrated in the context of role modeling.

When conflicting assumptions are built on both ends of a whole-part relationship, they entail OSP. Consider that an object A implements an operation by forwarding it to another object B which is nominally part of A. A may make coding assumptions that B is its part, and may hold that state in B does not undergo spontaneous changes. Further, A might even override some of the methods of B to ensure that it can filter or update its own state appropriately. However, direct calls to object B can be made if it has a different identity from A, which may invalidate the assumptions built into A. This results in bugs that are hard to find. In [25], Adapter pattern is cited to have this symptom.

In Figure 1, let us consider only the Conference Associate subrole and Reviewer subrole parts of the figure for ease of understanding about this symptom. If we make assumption that overriding is safely allowed, there is no problem since m3() calls m4() and not m2(). Indirect call m3() on OOPSLA will work well, if m3() is sent to OSP_paper. However, if message m3() is sent to OSP_paper and ConferenceAssociate role is modified such that m3() calls m2() and not m4(), broken delegation problem arises. In this case, this is an offshoot of the assumption. Initially, the assumption of safe overriding was made positively. As OOPSLA underwent spontaneous change, the assumption of safe overriding became invalid. Here, broken assumptions is a symptom that is an offshoot of broken delegation. This kind of situation in role modeling is OSP.

4.5 Broken Consultation

Broken consultation is another kind of misinterpretation of messages that leads to OSP. For instance, consider that an object A implements an operation by forwarding it to another object B. In case the object B makes an operation call on itself, it finds the definition of the adapter in the class associated with A, and not in the class associated with B. Such wrong semantics of message processing when method implementation specified in A has been overridden is broken consultation.

In Figure 1, if message m1() sent to OSP_paper object is forwarded to the object F of class Faculty wherein m1() calls m2() and if m2() of OSP_paper is made use of during the execution of m1() and not m2() of F, it is broken consultation. In this case, broken consultation appears as delegation. However, they are different. To illustrate, if the method holder F uses m2() of OOPSLA (its nearest object in the delegation path) in the course of execution of m1(), it is also broken consultation. Thus, broken consultation has broader semantics than delegation.

5. CONCLUSIONS

Is-role-of inheritance is presented as the kind of reuse mechanism suitable for an RPCM design. Is-role-of inheritance model necessarily involves OS. OS is not a problem. Different OSP symptoms that may arise in modeling the is-role-of inheritance have been investigated. Modeling is-role-of inheritance is bound to lead to some OSP symptom or the other. For instance, there are many ways in which the well known "self problem" can definitely manifest as broken delegation symptom or broken consultation symptom when a message forwarding mechanism such as delegation or consultation is used in modeling is-role-of inheritance. Henceforth, OSP can not be altogether eliminated in the new inheritance model. The existing role models mainly provided inheritance by delegation for method lookup but do not have strong claims as being OSP-free.

6. REFERENCES

[1] Antero Taivalsaari. Delegation versus Concatenation or Cloning is Inheritance too. *ACM Press OOPS Messenger*, 6(3):2–49, July 1995.

[2] Antero Taivalsaari. On the Notion of Inheritance. *ACM Computing Surveys*, 28(3):438–479, September 1996.

[3] M.Sakkinen. Disciplined Inheritance. In *Proceedings of ECOOP'89*, pages 39–56, 1989.

[4] William Harrison and Harold Ossher. Subject-Oriented Programming (A Critique of Pure Objects). In *Proceedings of OOPSLA*, pages 411–428, 1993.

[5] Alan Snyder. Encapsulation and Inheritance in Object-Oriented Programming Languages. In *Proceedings of OOPSLA*, September 1989.

[6] Peter Wegner and Stanley B. Zdonik. Inheritance as an Incremental Modification Mechanism, or What Is and Isn't Like. In *ECOOP '88*, pages 55–77, Oslo (Norway), 1988. Springer-Verlag.

[7] Lieberman H. Using Prototypical Objects to Implement Shared Behaviour in Object Oriented Systems. In *Proceedings of OOPSLA*, October 1996.

[8] Peter Wegner. Dimensions of Object-Based Language Design. *OOPSLA Proceedings in ACM Sigplan Notices*, pages 168–182, October 1987.

[9] Lynn Andrea Stein. Delegation is Inheritance. *OOPSLA Proceedings in ACM Sigplan Notices*, 22:138–146, October 1987.

[10] Klaus Ostermann and Mira Mezini. Object-Oriented Composition Untangled. *OOPSLA Proceedings in ACM Sigplan Notices*, pages 283–299, 2001.

[11] Daniel Bardou and Christophe Dony. Split Objects: A Disciplined Use of Delegation within Objects. In *Proceedings of OOPSLA*, pages 122–137, 1996.

[12] Edward Sciore. Object Specialization. *ACM Transactions On Information Systems*, 7(2):103–122, 1989.

[13] George Gottlob, Michael Schrefl, and Brigitte Röck. Extending Object-Oriented Systems with Roles. *ACM Transactions on Information Systems*, 14(3):268–296, July 1996.

[14] Barbara Pernici. Objects with Roles. In *IEEE/ACM Conference on Office Information Systems ACM SIGOIS*, number 1, pages 205–215, Cambridge, Massachutes, April 1990.

[15] Janaki Ram D., Ramakrishnan R, Srinivas Rao Ch., and Vivekananda N. CO-IP: An Object Model for Office Information Systems. In *Proceedings of the Third International Conference on Object Oriented Information Systems (OOIS'96)*, pages 31–43, London, December 1996.

[16] Bent Bruun Kristensen. Object Oriented Modeling with Roles. In *Proceedings of the Second International Conf. on Object Oriented Information Systems(OOIS)*, pages 57–71, Dublin, Ireland, 1995. Springer Verlag.

[17] Chandra Sekharaiah K., Janaki Ram D., and Kumar A.V.S.K. Typehole Model for Objects with Roles in Object Oriented Systems. In *Fourteenth Europeon Conference on Object Oriented Programming, Workshop Reader, LNCS 1964*, pages 301–302, Sophia Antipolis, France, June 2000. Springer Verlag.

[18] Chandra Sekharaiah K., Arun Kumar, and Janaki Ram D. A Security Model for Object Role Database Systems. In *International Conference on Information and Knowledge Engineering (Accepted, to appear)*, Las Vegas, Nevada, USA, June 2002.

[19] Joel Richardson and Peter Schwarz. Aspects: Extending Objects to Support Multiple, Independent Roles. In *Proceedings of the ACM SIGMOD Int. Con. on Management of Data*, volume 20, pages 298–307, May 1991.

[20] Albano A., Ghelli G., and Orsini R. Fibonacci : A Programming Language for Object Databases. *The VLDB Journal*, 4(3):403–414, 1995.

[21] G.Kappel, W.Retschitzegger, and W.Schwinger. A Comparison of Role Mechanisms in Object-Oriented Modeling. In *Proceedings of Modellierung'98, K. Pohl, A. Schurr, G. Vossen (eds), Bericht Nr. 6/98-I, Angewandte Mathematik und Informatik, Universitat Munster*, pages 105–109, 2001.

[22] Raymond K. Wong. Heterogeneous and Multifaceted Multimedia Objects in DOOR/MM: A Role-Based Approach with Views. *Journal of Parallel and Distributed Computing*, pages 251–277, March 1999.

[23] Günter Kniesel. Type-Safe Delegation for Run-Time Component Adaptation. In *ECOOP*, Lisbon, Portugal, 1999.

[24] Roel Wieringa and Wiebren DeJonge. The Identification of Objects and Roles - Object Identifiers Revisited. In *Technical Report IR-267*, Vrije Universiteit, Amsterdam.

[25] IBM Research: Subject-oriented Programming Group. Subject-Oriented Programming and Design Patterns. http:// www.research.ibm.com/sop/.

[26] Richard Helm, I. M. Holland, and D.Gangopadhyay. Contracts: Specifying behavioral compositions in object oriented systems. In *Proceedings of ECOOP/OOPSLA'90*, volume 1, pages 169–180, Ottawa, June 1990.