Objects and Roles: Modelling based on the Dualistic View

Tetsuo Tamai
(University of Tokyo)

July 3, 2002

1 Dual Concepts of Objects and Roles

Software engineering has been producing a number of models or modeling methods, e.g. the data flow model, the state transition model, and the entity relationship model, just to name a few [35]. Models are in general composed of elements and structures combining those elements. Thus, the problem of constructing a "good" model depends on how to choose an appropriate set of model elements and how to identify a meaningful structure composed of those elements.

Models are developed mainly for two objectives. One is to map a real world or domain to a model so as to enable analysis on the problem within its context. Domain models and analysis models fall in this category. The other objective is to build a system image to be developed as a model. System models and design models are examples of the latter.

Elements of system design models have traditionally been called modules and the process of identifying modules is called modularization. Ever since object-oriented technology was widely recognized as applicable not only to programming but also to analysis and design, objects have been adopted as the elements of models. Now, object models are being used "seamlessly" from the analysis phase through the design phase and to the implementation phase.

When modeling process, regarded as an activity of identifying objects and their structures, is conducted in the top-down manner, it can be called decomposition and when conducted in the bottom-up manner, it can be called composition. Although the terms composition and decomposition have been used for a long time in conventional modularization techniques, they are still valid in the context of object-orientation. Actual modeling is not a pure composition or decomposition process but some mixture of the two. But still it should be correct to say that the major task of modeling is achieved by way of composition and decomposition.

However, there have been arguments that the monistic view of modeling based solely on objects is often insufficient. Instead, a dualistic view based on objects and roles are advocated by a number of researchers and practitioners. Motivations for introducing roles are various and relations put between objects and roles are also different by models. One way of classifying modeling methods based on the object/role dualism is by the weight of balance put on objects and roles.

1) Objects-are-main-and-roles-are-sub case: In major OO analysis methods, roles are considered at the stage prior to the step of identifying a set of objects, where roles represent responsibilities to be assumed by some objects. Thus, listing up roles is a means for listing up objects. This type of roles are considered in terms of collaborations by objects and appear only at the analysis or general design phase, disappearing from implemented programs.

2) Objects-and-roles-are-equally-treated case: This may be called a genuine dualism while the first case is an unbalanced dualism. Many computational models and languages that fall in this category have been proposed, most of which are for research purposes.

The second dimension of classifying dualistic modeling methods is by motivations, including collaboration, adaptation, evolution, and mobility. In the following sections, we will survey modeling methods based on the object/role dualism, characterizing them by their motivations.

2 Motivations

2.1 Collaboration

As mentioned in the previous section, collaboration of objects is treated in many OO analysis methods in terms of roles. In this view, collaboration is interpreted as functions or behaviors realized by interaction among a group of roles. Objects are entities independent from roles and an object participates in a collaboration by playing some role of the corresponding collaboration pattern. An object may take part in multiple collaborations assuming different roles in different collaborations. Thus, the characteristics of an object may be clarified by consolidating roles the object plays in multiple collaborations.

A typical way of describing a collaboration is by specifying use-cases or behavioral scenarios as observable behaviours of the collaboration. This approach was originally advocated by I. Jacobson [19] and is inherited by the Unified Modeling Process [18]. In UML, use-cases are illustrated by use-case diagrams and also sequence diagrams or collaboration diagrams are used in combination to describe collaborations [2]. A similar approach is taken by Wirfs-Brock et al. [37], where responsibilities to be taken by objects for achieving various types of collaborations are given much attention and a kind of Class-Responsibilities-Collaboration Cards [1] is adopted to describe these relations. In these methodologies, the word *role* is not used but the corresponding concept is captured as an aspect of objects engaged in collaborations [1]. Thus, collaborations are considered for obtaining an appropriate set of objects that compose an object model and responsibility is not an orthogonal concept to object but its granularity is close to methods of an object.

[1]In UML, the word *role* is used for a different meaning, i.e. to give a name to each direction of an association.

In some other OO development methodologies, the concept of roles is given much higher position so that the term “role modeling” is created and extensively used. A typical example is the OOram methodology [26], which not only defines role models but also integrates them by the step of role model synthesis. Design patterns can also be regarded as describing patterns of collaboration. The works of D. Riehle further extend this view and employ the notion of role modeling to model object migration [36] and to design composite patterns [27] and frameworks [28]. Also related is the work by B. Kristensen et al. [21].

Also related is the notion of *contracts*. Contracts, proposed by R. Helm et al. [14] is a construct for the explicit specification of behavioral compositions. A contract defines a set of communicating participants and their contractual obligations. This notion of *participants* correspond to roles but participants are actually objects and thus the separation of objects and roles are somewhat blurred.

In these methodologies, roles play an important part at the phases of analysis and design but usually become invisible in the implementation. However, there are some works that aim at preserving roles explicitly in programs. For example, VanHilst and Notkin [34] used class templates of C++ to implement roles. One of the objectives of this proposed method is to reuse roles besides or even in stead of objects.

2.2 Adaptation, Evolution and Mobility

As the case of collaborations shows, there is a reciprocal relation between objects and roles, i.e. an object can play multiple roles at a time and a role can be played by multiple objects. This scheme can be conveniently employed in various situations. One of the straightforward application is to handle objects that share properties of different classes. M. Fowler [7] gives an example of personnel roles in a company to be assumed by employees. He lists up engineers, salesmen, directors and accountants as roles and put a question how to deal with situations such that a person plays more than one role or a person changes his or her role in the lifetime. He shows several patterns that solve this problem and gives a generic name *role pattern*.

2.2.1 Adaptation

Fowler’s motivation can be paraphrased or slightly extended as needs for adaptation. This is a motivation to let objects adapt to multiple environments or changing environments. Objects endowed with adaptability are often called agents and pursued not only in the software engineering community but also in the AI and network communities [3]. One of the key issues is to realize adaptation dynamically.

Honda et al. [15] gives an example of adaptation. A woman Hanako, modeled as an object, marries with Taro and adapts to the environment *family*. She then gets employed as a researcher by a research laboratory and adapts to the environment *laboratory*. The adaptation should be made dynamically and the object Hanako should preserve its identity when she enters a new environment like the lab or even after she quits the lab for some reason.

There can be various ways to tackle this problem. There are efforts, especially in the field of distributed AI or multi-agent systems, to develop technologies that let objects (or agents) autonomously transform or re-organize themselves according to the situation or environment they are in. A typical example is Gaea, an “organic programming language” designed by Nakashima et al. [25]. A program in Gaea composed of units called *cells* adapts itself to the situation through re-ordering and replacing cells dynamically. Approaches like this is challenging but still under development. On the other hand, an approach based on multiple inheritance or a mixin type technique is already matured but, as easily guessed, hard to meet the requirement of dynamic change. Fowler shows some neat techniques including “hidden delegate” and “state object” to implement dynamic role assignment within the conventional object-oriented programming framework.

Of course, the delegation based approach is a natural alternative [33]. However, delegation is a relatively primitive operation. We may want to think at a more abstract level. A simple but effective way is provided by the object/role dualistic model. An environment is represented by a set of roles that interact each other. The environment itself may also be provided with its own attributes and methods. An object adapts to the environment by assuming or combining with one of its roles and thus acquiring properties and capabilities to fit in the new environment.

Honda et al.’s work on *Morphe* [15] can be seen as an example based on this role/environment approach. In their model, suppose an object (e.g. Hanako) enters an environment (e.g. a laboratory) and assumes a role (e.g. a researcher), then the object acquires a new set of attributes and behaviors or alters some of the attributes and behaviours already possessed by the object through following transformation rules associated with the role. Why this strategy of employing transformation rules is adopted in their model may be partly explained by the fact that the underlying language is a constraint based OO language.

Ubayashi & Tamai’s Epsilon model [32] takes a more straightforward approach of binding an object to a role instance when an object enters a context (used here as a synonym for an environment). More details of this binding mechanism will be explained in the later section.

2.2.2 Evolution

When adaptation takes place dynamically in the course of time, it can be called evolution. In this view, objects evolve as the environment surrounding them changes. Gottlob et al. [10] deals with dynamic change of objects (but since their main concern is data base, objects are more like data base schemas) using the concept of roles. They claim that inheritance is class based and thus inconvenient for handling dynamic changes. Instead, they propose a role hierarchy and realize specialization and inheritance at the instance level.

They list six features of roles:

- Various roles of an entity may share common structure and behaviour;
- Entities can acquire and abandon roles dynamically;
- Roles can be acquired and abandoned independently of each other
- Entities exhibit role-specific behaviour;

- Roles restrict access to a particular context;
- Entities may occur repeatedly in the same type of role.

This is a good summary of the properties of roles that hold not only in their model but also in other role based models.

2.2.3 Mobility

When adaptation takes place geographically, i.e. an object adapts to a new environment which is located away from the old place it belonged to and where the object has moved into, then it can be called adaptation due to mobility. Kumeno et al.'s Flage [22] can be seen as a model of realizing mobile agents through such adaptation. An object (or agent) enters a new possibly remote environment and adapts to it through acquiring necessary functions. In their model, the concept of roles is not employed and every object that enters an environment acquires the same set of behaviours. If behaviours can depend on roles that compose the environment, then the model becomes close to Epsilon.

There are a number of researches and developments of mobile agent systems [23, 17, 29, 30] but no other approaches as far as we know take the concept of object/role dualism.

2.3 Other related motivations

There are many other works that by and large share motivations as described above but take different approaches for solutions. Notable ones are subject oriented programming and aspect oriented programming. They share the notion that models or systems can be grasped differently by views. The former calls the view *subject* and the latter *aspect*.

Subject Oriented Programming Harrison & Ossher [13] states that their goal is "to facilitate the development and evolution of suites of cooperating applications." They specifically emphasize that the same object would be seen differently by "subjects" and yet there should be coherent intrinsic properties inside the object. They propose some probable methods for reconciling various views. The way they see cooperation in this framework is by sharing an object and explicit collaborations as discussed in Section 2.1 are not necessarily intended.

Aspect Oriented Programming Kiczales et al. [20] claims that a system modularization structure designed from one aspect is often in conflict with modularization from another aspect. Thus, they propose a method of describing aspects separately and then weaving them together to obtain a consolidated system. An aspect here may correspond to a field of collaboration but aspects are designed at the programming phase and not created dynamically. Thus, the aspect oriented programming is not so conscious about adaptation or dynamic evolution but rather focuses on the modularization methodology.

3 Composition as an Implementation Mechanism

As we have seen briefly in Section 2.2.1, there can be at least three different approaches in implementing object/role relationships:

1. multiple inheritance,
2. composition,
3. transformation.

Multiple inheritance has a problem in handling dynamic changes. Transformation, especially that of exploring autonomous adaptation, is still a premature technology. Compared to them, composition is a simple but powerful approach that suits for dynamic adaptation.

In the conventional OO methodology, composition has also been treated statically, i.e. a composite retains references to its components in its instance variables. References may be changed through assignment to the variables but the structure of the composite does not change and the assignment is constrained by types and other interface conditions. Delegation based languages like Self allow more flexible composition. But they mainly focus on how to share common functions at the instance level and have little intention of letting two or more objects combine together to form a larger structure. Delegation is one directional and two objects related by delegation relation preserve their relatively independent nature.

How composition is used in the works surveyed so far? Honda et al. calls their mechanism in Morphe adaptive composition but as explained before, their composition is accompanied by transformation. It is generally not easy to decompose a composite if the composition is undertaken through transformation.

Gottlob et al. [10] does not use the word composition. In their model, a set of roles set up a hierarchy just like a class hierarchy but it is explained "a subtype in a role hierarchy does not inherit definitions of instance variables and instance methods from the supertype. (...) inheritance is defined at the instance level rather than at the class level." This sounds quite complicated but things may get simpler if it is explained in terms of composition. An object instance can be combined to a role instance by composition, where the role hierarchy tree specifies the route of delegation at the instance level. This delegation hierarchy could also be interpreted by composition, i.e. a role at a node of the tree can be regarded as a composite packing nodes along the path from the role node to the root.

Contracts by Helm et al. use the word compositions to describe their mechanism. A contract specifies a composition at the class level and by instantiating a contract, a behavioral composition is created. In that sense, compositions of the contracts model remain in the conventional OO framework.

Compositions of the Epsilon model has the following characteristics:

1. Composition takes place when an object instance and a role instance are bound together;
2. An object instance can be bound to multiple role instances residing in different contexts;

3. As a role is also an instance it has its own state as well as its own set of methods and preserve the state even after the separation from its pair object;

4. The state of an object and that of a role construct a Cartesian product state after composition;

5. A method of an object and a method of a role can be overridden or renamed by another method of the counterpart role or object and thus interaction between the object and role states is made possible;

6. The above mechanism indicates that the binding of an object and a role can be bi-directional as opposed to the uni-directional relation of delegation.

Behavioral semantics of such composition appear to be intuitively clear but still require rigorous treatment. Statechart [11] has a construct of AND composition of two or more state machines. Semantics of Statechart have been given by many researchers, a typical one specified by the originator [12].

Also related are compositions of labeled transition systems. Behavioral properties of composed systems where some states are possibly hidden to the observer can be analyzed [4, 5].

Another approach for analyzing behavioral properties of composite systems is proposed by S. Iida [16]. His approach is based on hidden sort algebra devised by J. Goguen [9] and implemented in CafeOBJ, an algebraic specification language [6].

4 Epsilon Model and Language

In this section, we briefly overview the Epsilon model and language that is based on the object/role framework.

4.1 Epsilon Model

The Epsilon model consists of three kinds of entities: *objects, roles and contexts*. Each role belongs to a unique context. A context is a field of collaboration or an environment and roles belonging to the context collaborate each other within that context. A context defines a scope of reference for roles, i.e. a role can communicate only with other roles in the same context. Thus, contexts provide description units of collaborations and realize the separation of concerns.

An object can be bound to a role, through which it participates in the collaboration specified by the context the role belongs to. This binding mechanism is basically what is explained as composition in the section 3 and thus realizes adaptation of objects. Binding takes place dynamically and an object can be bound to different roles at a time. Also, it can unbind itself from the role any time. As opposed to an object, a role cannot be bound to another role.

A context is essentially an object in that it may have its own state (attributes) and methods. As a result, a context can be bound to a role of another context and thus a layer structure of contexts can be created.

Figure 1 illustrates an example of Contract Net Protocol [31]. It is a protocol to solve a problem collaboratively through negotiation of multiple processing nodes. A contract may be given by a manager to a contractor who has bidden the lowest price. A node can be a manager of one contract and a contractor of another. This problem can be conveniently modelled by Epsilon; a contract is represented by a context and a manager and contractor(s) by roles.

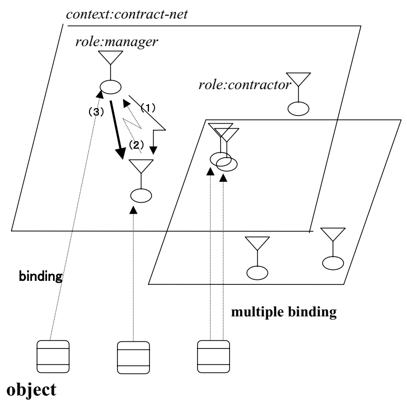

Figure 1: Contract Net Protocol

4.2 Epsilon Language

The language based on the Epsilon model is also named Epsilon. In Epsilon, roles and contexts are first class constructs. Roles and contexts as well as objects are defined in programs as classes and instantiated at the run time. Binding of objects to roles is always done at the instance level.

As collaborations are explicitly described and encapsulated as contexts in Epsilon, they can be reused as program components. Thus, design patterns of Gamma et al. [8], for example, are good targets for building reusable program components and they will be used not just as a catalogue of design know-how's but reusable components.

A preliminary version of an Epsilon compiler was implemented on ABCL/R3, a reflective concurrent object-oriented language [24].

References

[1] K. Beck and W. Cunningham. A laboratory for teaching object-oriented thinking. In *OOPSLA '89*, pages 1–6, 1989.

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

[3] J. M. Bradshaw. *Software Agents*. The MIT Press, 1997.

[4] Shing Chi Cheung and Jeff Kramer. Context constraints for compositional reachability analysis. *ACM Transactions on Software Engineering and Methodology*, 5(4):334–377, October 1996.

[5] Shing Chi Cheung and Jeff Kramer. Checking safety properties using compositional reachability analysis. *ACM Transactions on Software Engineering and Methodology*, 8(1):49–78, 1999.

[6] Ruazvan Diaconescu and Kokichi Futatsugi. *CafeOBJ Report: The Language, Proof Techniques, and Methodologies for Object-Oriented Algebraic Specification*. World Scientific, 1998.

[7] Martin Fowler. Dealing with roles. http://www2.awl.com/cseng/titles/0-201-89542-0/apsupp/. supplemental information to *Analysis Pattern*, Addison-Wesley, 1997.

[8] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[9] Joseph Goguen and Grant Malcolm. A hidden agenda. Technical Report CS97-538, UCSD, April 1997.

[10] G. Gottlob, M. Schrefl, and Röck. Extending object-oriented systems with roles. *ACM Transactions on Information Systems*, 14(3):268–296, July 1996.

[11] David Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8:231–274, 1987.

[12] David Harel and Amnon Naamad. The statemate semantics of statecharts. *ACM Transactions on Software Engineering and Methodology*, 5(4):293–333, 1996.

[13] W. Harrison and H. Ossher. Subject-oriented programming (a critique of pure objects). In *OOPSLA '93*, pages 411–428, 1993.

[14] R. Helm, I. M. Holland, and D. Gangopadhyay. Contracts: Specifying behavioral compositions in object-oriented systems. In *ECOOP/OOPSLA '90 Proceedings*, pages 169–180, October 1990.

[15] Yasuaki Honda, Shigeru Watari, and Mario Tokoro. Compositional adaptation: A new method for constructing software for open-ended systems. *Computer Software*, 9(2):122–136, 1992. in Japanese.

[16] Shusaku Iida. *An Algebraic Formal Method for Component Based Software Developments*. PhD thesis, Japan Advanced Institute of Science and Technology, 1999.

[17] ObjectSpace Inc. Objectspace voyager technical overview. http://www.objectspace.com/voyager.

[18] I. Jacobson, G. Booch, and J. Rumbaugh. *The Unified Software Development Process*. Addison-Wesley, Reading, 1999.

[19] I. Jacobson, M. Christerson, P. Jonsson, and G. "Overgaard. *Object-Oriented Software Engineering: A Use Case Driven Approach*. ACM press, 1992.

[20] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C.V. Lopes, J.-M. Loingtier, and J. Irwin. Aspect-oriented programming. In *Proceedings of the European Conference on Object-Oriented Programming(ECOOP), Finland*. Springer-Verlag, June 1997.

[21] Bent Bruun Kristensen and Kasper Osterbye. Roles: Conceptual abstraction theory and practical language issues. *Theory and Practice of Object Systems*, 2(3):143–160, 1996.

[22] F. Kumeno, H. Sato, T. Kato, and S. Honiden. Flage: A programming language for adaptive software. In *Proceedings of ICSE'98*, volume 2, pages 103–108, 1998.

[23] B. D. Lange and M. Oshima. *Programming and Deploying Java Mobile Agents with Aglets*. Addison-Wesley, 1998.

[24] Hidehiko Masuhara, Satoshi Matsuoka, and Akinori Yonezawa. Implementing parallel language constructs using a reflective object-oriented language. In *Reflection Symposium '96*, pages 79–91, April 1996.

[25] H. Nakashima and I. Noda. Dynamic subsumption architecture for programming intelligent agents. In *Proc. of International Conf. on Multi-Agent Systems 98*, pages 190–197. AAAI Press, 1998.

[26] T. Reenskaug, P. Wold, and O.A. Lehne. *Working with Objects: the OOram Software Engineering Method*. Manning Publications, Greenwich, 1996.

[27] D. Riehle. Composite design patterns. In *OOPSLA '97*, pages 218–228, Oct. 1997.

[28] D. Riehle and T. Gross. Role model based framework design and integration. In *OOPSLA '98*, pages 117–133, Vancouver, Oct. 1998.

[29] K. Rothermel, editor. *Mobile Agents*, volume 1447 of *Lecture Notes in Computer Science*. Springer, 1998.

[30] I. Satoh. Agentspace web page. http://islab.is.ocha.ac.jp/agent/index.html.

[31] R. G. Smith. The contract net protocol: High-level communication and control in a distributed problem solver. *IEEE Trans. on Computers*, 29(12):1104–1113, 1980.

[32] Naoyasu Ubayashi and Tetsuo Tamai. Modelling collaborations among objects that change roles dynamically and its modularization mechanism. *IEICE Trans. on Information and Systems*, 1999. to appear.

[33] David Ungar, Craig Chambers, Bay-Wei Chang, and Urs H"olzle. Organizing programs without classes. *Lisp and Symbolic Computation*, 4(3):37–56, 1991.

[34] M. VanHilst and D. Notkin. Using Role Components to Implement Collaboration-Based Designs. In *OOPSLA '96*, pages 359–369, 1996.

[35] R. Wieringa. A survey of structured and object-oriented software specification methods and techniques. *ACM Computing Surveys*, 30(4):459–527, 1998.

[36] R. Wieringa, Wiebren de Jonge, and Paul Spruit. Using dynamic classes and role classes to model object migration. *Theory and Practice of Object Systems*, 1(1):61–83, 1995.

[37] R. Wirfs-Brock, B. Wilkerson, and L. Wiener. *Designing Object-Oriented Software*. Prentice Hall, Englewood Cliffs, 1990.