

ROLES VERSUS CLASSES

Haibin Zhu, *Senior Member, IEEE*

*Department of Computer Science and Mathematics, Nipissing University,
100 College Dr., North Bay, ON, Canada, P1B 8L7*

haibinz@nipissingu.ca

Abstract

To improve object-oriented technologies, roles are proposed to support separation of concerns, object collaboration, and object evolutions. Compared with classes, there is confusion regarding roles in modeling, because classes are a well-accepted concept and mechanism of object-oriented technologies while roles are not. This paper explores the nature of classes and roles, clarifies the similarities and differences between them, compares the advantages and disadvantages of them in different aspects, proposes that a programming language should accommodate roles at the same level as classes, and points out the existing challenges in applying roles in programming languages.

Keywords: *Object-Orientation, Modeling, Classes, Roles*

1. Introduction

Classes are the key concept and mechanism in object-oriented technologies. From the viewpoint of programming, we could find that the major difference between an object-oriented programming language and a traditional or procedural one is that there is a new key word *class* applied. Classes show the advances of object-oriented programming clearly. With the introduction of classes, the major tasks of object-oriented programming are to identify and design classes, create objects by instantiating classes and send messages to objects.

Roles are commonly applied concepts in many fields, such as sociology, psychology, social psychology, behavioral science, management, and drama. In our society, roles and classes are similar concepts used to describe the phenomenon of people's community and relationships. Roles can be used to map people's natural organizations, task distributions, system analysis, system design and system construction. They have been deeply discussed in data modeling and system design. However, there is a largely unexplored question: what exactly are the differences between classes and roles? Because of this unanswered question, there are many arguments surrounding the applications of roles in system modeling, analysis, design and implementations. For example, some express roles with classes [10]; some with instances [1]; and some even with metaclasses [7].

In other words, although there are many discussions about roles, there is an evident lack of comparison between classes and roles. Classes have contributed to modeling and

programming with their fully-developed characteristics but roles have not. Apparently, the role concept and the class concept cannot replace each other; they have their own advantages. The problem is that the advantages of roles have not been developed thoroughly. Because there are many different concerns about roles, we mainly concentrate on roles in system modeling, analysis and designs.

In this paper, the author analyzes the characteristics of classes and roles, and compares their similarities and differences in modeling and design. This clarification guides the applications of roles into programming and modeling.

This paper is arranged as follows. Section 2 reviews the concept of class and clarifies the characteristics of classes; Section 3 reviews the major characteristics of roles; Section 4 describes a tool to extract roles from a class; Section 5 discusses the similarities and differences between them, Section 6 concludes the paper and proposes the topics for future research.

2. Classes

Classes are a step-stone in the advancement of programming technologies. They make it possible to make object-oriented programs. Every object-oriented programming language must support the structure of classes. Classes are closely related to abstraction and classification [23].

Abstraction is a tool to know, create, and understand a complex thing. Abstraction is also a general methodology to study the world and set up our knowledge of it. When we have many concrete objects to control, we generally remember their common properties and each one's specialties. This is an abstraction process. The result from this process is an abstracted concept.

By abstraction, we can understand other relevant concepts such as information hiding. We can say that abstraction is a way to implement information hiding, or by information hiding, we obtain abstraction.

Abstraction is also a difficult ability for people to master. Just as what Confucius said more than 2000 years ago, “四十而不惑 (Si Shi Er Bu Huo)” that means he completely mastered the abstraction ability at the age of forty. Classes make it easy to abstract with formulated structures and a step-wise approach. In problem solving, if the objects in a problem are too complex to master, we need to make a new abstraction: a group of objects, i.e., a class.

Classification is also a general methodology that is applied in many domains of human lives. Some researchers even categorize classification as an abstraction [6]. Generally speaking, that is true. But in programming, classification and abstraction are different because they concentrate on different aspects of problem solving. An object-oriented system also takes it as a key mechanism to arrange elements in programming. Classification comes from the natural requirements to solve problems in any domain of application; to describe commonalities in collections of objects. It is the basic activity of people in organizing perceptions in their early mental development. It is the same way for scientists to organize knowledge within scientific disciplines, and for application programmers to organize domain knowledge and behaviors.

There is an idiom in China called “物以类聚，人以群分” (Wu Yi Lei Ju, Ren Yi Qun Fen) which means things of one kind come together and people are divided in groups. This idiom tells us the basic natural regulations of classification in the real world.

Therefore, classes are a mechanism of classification and a tool of abstraction. In other words, a class is either a classification or an abstraction or both of objects. A class is not only a concept, but is also a very practical mechanism in a program. From the abstraction viewpoint, extracting classes from concrete objects is definitely a process of an abstraction.

A class serves two distinct purposes in general. It is a definition of an implementation (methods and data structures) shared by a group of objects and it is a template from which objects may be created. It contains a definition of the state descriptors and methods for an object. There are two views for a class: narrow and broad.

The narrow view of a class is that objects are instances of a class. Their state and behavior structures are determined by the class definition. A class is a template to create objects.

The broad view of a class is that a class consists of an object-warehouse or object-factory. Object warehouse means that a class implicitly maintains a class extent. A class extent consists of all instances of one class, like a set. Hence, an object can be "of" a certain group of objects. An apple is of Fruit, a pear is of Fruit, but they are not instances of Fruit. Classes in this meaning are sets. Object-factory means that there exists a constructor for each class. The purpose of this constructor is to generate new instances of a class.

In general, a class provides a definition of the structure of its instances. It defines the names of attributes (state) and methods (behavior) of an object belonging to this class. Object-Oriented programming is actually programming with classes.

A *class* is a user-defined type. It is defined as a set of objects that share a common structure and common behavior [5]. A *class* can be defined as a quadruple [23], $c ::= \langle n, \mathcal{D}, \mathcal{F}, \mathcal{X} \rangle$, where

- n is an identification or a name of the class;
- \mathcal{D} is a space description for memory;

- \mathcal{M} is a set of the method specifications and implementations; and
- \mathcal{X} is a unified interface for all the objects of this class and it includes all the method specifications.

To express inheritance among classes, we can expand the definition of classes to subclasses, a subclass is defined as $c_s ::= \langle n, [c], \mathcal{D}, \mathcal{F}, \mathcal{X} \rangle$, where

- $n, \mathcal{D}, \mathcal{M}$, and \mathcal{X} have the same meanings as those in the definition of classes; and
- $[c]$ is a set of classes that are called super classes of this class.

In the construction of an object-oriented program, we create many classes and arrange them in well-defined structures. Classes may have many relationships, such as, use, composition, and inheritance.

A use relationship exists between two classes that need to communicate. One class may use one or more classes in its methods to send service requests or messages. This is a major relationship in an object-oriented system.

An inheritance relationship exists between two classes that belong to a classification route. One class inherits one or more other classes to share the properties. A set of subclasses inherit and specialize their superclass(es).

Classes may have composition relationships, i.e., one class may use other classes to describe its data structures. In other words, the instance of one class may be composed of the instances of other classes.

We can find that many properties of object-orientation are from the class concept.

- Abstract data types: A class is a user-defined type in a program.
- Inheritance provides further data abstraction: It supports both sharing and classification.
- Encapsulation and information hiding: Their use leads to easier and less error-prone product development, and easier maintenance.
- Procedural abstraction based on data abstraction: By this mechanism, we can make information hiding a real thing. This is a revolution compared with traditional procedural programming. Traditional procedure abstraction cannot totally implement information hiding.

3. Roles

In common sense, the term “role” is derived from the theater and refers to the part played by an actor. A role represents a specific status that possesses certain rights and accompanying responsibilities. It can be defined as a set of regulations defining what the behavior of a position member should be. A role defines a set of responsibilities and capabilities needed to perform the activities relevant to these responsibilities [2]. A role can also be defined as the prescribed pattern of behavior expected of a person in a given situation by virtue of the person’s position in that situation [4]. It is simply defined as a position in a social structure. A position means a more or less institutionalized or commonly expected and understood designation in a given social structure

such as an accountant, mother, or church member [2]. A role is a set of expectations about behavior for a particular position within a work system. Generally speaking, a role is a position occupied by a person in a social relationship. Occupying this position, one possesses special rights and takes special responsibilities.

Roles are defined as a concept that is founded but not semantically rigid [12]. “Founded” means that a concept can exist essentially independently. “Semantically rigid” means that a concept contributes to the identity of its instances. This definition can help determine if a concept is a role. Objects that depend on other objects for their existence should not be roles. Roles allow not only for the representation of multiple views of the same phenomenon, but also for the representation of changes in time. Roles are also the bridges between different levels of detail in an ontology structure.

Many researchers consider roles as a fundamental concept in modeling [1, 3, 9, 12, 13, 17]. Roles are entities that temporarily confine the behavior of objects. A role is considered as an abstraction and decomposition mechanism related to objects. Objects may play roles. When an object plays a role, it accepts messages and provides services related to its role. A role constitutes a part of an object’s behavior that is obtained by considering only the interactions of that role and hiding all other interactions.

There are many different discussions about the concepts of roles. Some aspects of roles seem contradictory. The comprehensive studies on roles as fundamental concepts and mechanisms [1, 3, 9, 12, 13, 17] are good guidelines for understanding roles in object systems. To support role playing with object-oriented programming languages, we need to confine our role concepts in the sense of data modeling and programming. Based on the basic view of object-orientation and the system modeling requirement, we can constitute the fundamental principles relevant to object evolution and separation of concerns based on roles.

When an object plays a role, it accepts messages and provides services related to its role. A role constitutes a part of an object’s behavior that is obtained by considering only the interactions of that role and hiding all other interactions. The following characteristics of roles are commonly accepted in the literature of modeling methodologies:

- 1) Roles are a concept or specialization of a concept.
- 2) Roles can be acquired and abandoned independently of each other.
- 3) Roles can be organized in hierarchies, generalized or specialized.
- 4) Roles model a perspective on a phenomenon.
- 5) Roles are bound to an existing object.
- 6) Roles are used to emphasize how entities interact with each other.
- 7) Roles can be dynamic, involving sequencing, evolution, and role transfer.
- 8) An object may play several roles at a time.
- 9) Various roles of an object may share common structure and behavior.

10) The states and features of an object can be role-specific.

To express the evolution of objects in object-oriented programming [1, 10, 7], roles are considered as states and tags. The major concern is how to organize the roles in a well-defined structure and let the objects attach their roles easily.

An *object role* shows the commonalities of a group of role objects. It can be defined as a class, $r_c ::= \langle n, \mathcal{D}, \mathcal{M}, \mathcal{X} \rangle$ where,

- n is an identification or a name of this object role;
- \mathcal{D} is a space description for memory;
- \mathcal{M} is a set of the method specifications and implementations; and
- \mathcal{X} is a unified interface of all the instances of this object role class and it includes all the method specifications.

In the sense of object role, there are many role instances for a role class. Role instances are attached to an object that is playing them. These kinds of roles can be well applied in expressing the evolution of objects. The state of an object role is a part of the state of the object that is playing the role. The requests are implicitly described in the implementation, i.e., \mathcal{M} of the object role.

From the viewpoint of object roles, services are the key concerns. In other words, every object, data item, or component has responsibilities to serve others and a role is a part of all the responsibilities of an object. That is to say, an object is considered as a server.

In system management, roles are taken as a mechanism to group human users to simplify the job of managing them. It is the basic idea from Role-Based Access Control (RBAC) [8, 16, 18] and database systems. The role concept from RBAC is now successfully applied in the industry such as system security administration and database systems. In system management, human users are the major concerns when designing roles. It is human users but not objects that play roles. Therefore, rights are the key concern of roles in RBAC. That is to say, in a system with RBAC, a user has rights to access the objects in the system and users play one role at a time when interacting with the system, i.e., possessing the rights relevant to this role. System administrators can assign rights to roles but not to users so as to save time, because a role can be assigned to many people. Note that the number of roles is greatly less than the number of users. That is why roles can be used to group people in the sense of access rights.

In a social environment, roles are taken as a tool to specify human behavior. The design of information systems should reflect the concerns of roles in a social environment. Current information systems have become a tool of collaboration. People generally interact with each other by firstly interacting with a computer or an information system (Fig. 1).

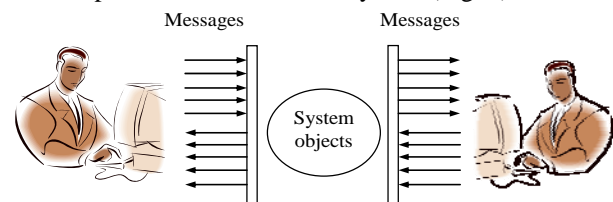


Fig. 1. People-System-People Interactions.

If a system is considered as composed of objects that collaborate with others, the objects should be taken as both servers and clients. Therefore, the roles in the system should represent both rights to issue requests and services to provide.

By accommodating both request and services, roles can be used to express different meanings or usage for objects, system components, systems and people, e.g., interfaces, interactions, specifications, transitions, evolutions and separation of concerns.

In reality, a role in a working environment is actually a wrapper with a service interface [19] and request interface [15], [21, 22, 24, 25]. A person's role can be divided into two parts: the service interface including incoming messages, and the request interface including outgoing messages.

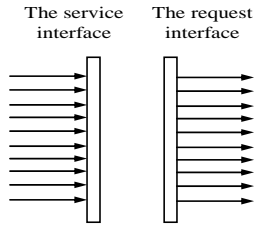


Fig. 2. A Role as a Wrapper of an Object.

From Fig. 2, we see the concept of an interface role. Roles are abstract entities to express the interfaces between role players. Roles only specify what the services and requests are. How they are processed depends on their players. In modeling, system analysis, and system design, roles belong to this category.

An *interface role* shows both the requests and services of objects. Incoming messages are used to express services and outgoing messages to express requests. It is defined as $r_i ::= \langle n, \mathcal{M}_{in}, \mathcal{M}_{out} \rangle$ where,

- n is the identification of the role,
- \mathcal{M}_{in} expresses a set of service specifications; and
- \mathcal{M}_{out} expresses a set of request specifications.

Interface roles are used in project management, system analysis and design when architecture and structures are more important. The identification and specification of interface roles are the major tasks for project managers, system analysts and systems designers. The interface role of an object can be extracted from the class of this object.

If we think of more practices, we find that only interface roles do not express all the meanings of natural roles. In reality, there are many roles that possess very concrete specifications of how to complete a task. For example, in a factory of electronics, the role of an assembly line worker is strictly specified as how to perform the operations on the line within a time limit. Similarly, in robotics, roles are specially designed processes that are put into the memory of a robot. When the robot plays this role, it actually behaves according to the process specified in the role [20].

Process roles are concrete descriptions in specifying the behaviors of their players. They specify not only what services and requests are but also how they are processed. In agent systems and process management [14], roles belong to this category.

A *process role* shows how to implement both the requests and services of objects. Incoming messages are used to invoke services and outgoing messages are used to issue requests. It is defined as $r_p ::= \langle n, \mathcal{D}, \mathcal{P}_{in}, \mathcal{P}_{out} \rangle$ where,

- n is the identification of the role,
- \mathcal{D} is a space description for memory to accommodate the state of this role;
- \mathcal{P}_{in} expresses a set of service implementations; and
- \mathcal{P}_{out} expresses a set of request implementations.

Process roles are used in system implementation when concrete jobs should be specified clearly and exactly. The design and implementation of process roles are the major tasks of programmers.

In fact, process roles can be applied to business management. That is to say, to play a role, a manager staff actually performs activities clearly specified by the role. Software engineering itself hopes to regulate such kinds of processes in the development of software. Therefore, process roles can be applied in process management or specifications of software engineering.

4. Extracting Roles From Classes

It is stated above that an interface role can be extracted from a class. That is to say, when classes are made with a programming language such as Java, both services and requests can be obtained. In traditional object-oriented programming, programmers ignore the request aspect in designing a class. It is taken for granted to be able to send requests to any class in the language concerned. After a class is designed, the requests are clearly confined in the class definition.

To accomplish this extraction task, we design and implement a specific tool that is to extract all the services and requests of a class. The aim is to know if a class (objects) is able to play a designated role specified by both requests and services. Because it is easy to extract the service interface of a class, we concentrate on the request interface in this section.

The value of this tool is based on the assumption of a well-defined role mechanism. Suppose there is such a mechanism, this tool can find an appropriate class, instantiate it, and have its instance to play a specified role. It can also find all the roles specified in a system an instance of a class can play. This is an important phrase in role-based development [22].

The tool is actually a simplified Java pre-compiler. It analyzes a class and extracts all the names of the classes or types and the methods used in the class. The class names are generally in statements of object declaration and a parameter list of a method invocation. The method names are typically in statements of message passing.

In order to accomplish this task, the tool completes the early actions of a compiler according to the Java grammar described in [11] and all types are fully resolved.

The result obtained from this tool is a list of method specifications that are required by the class. Note that the service list is easy to extract from a regular class. Fig. 3 shows a class and Fig. 4 shows part of the result.

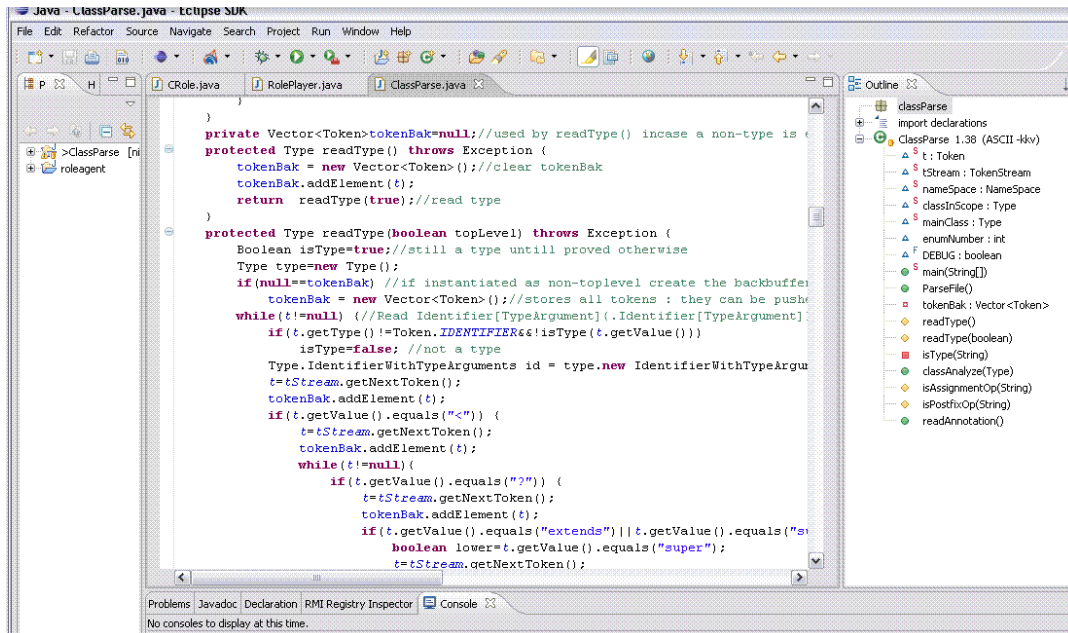


Fig. 3. The Example Class to be Extracted from.

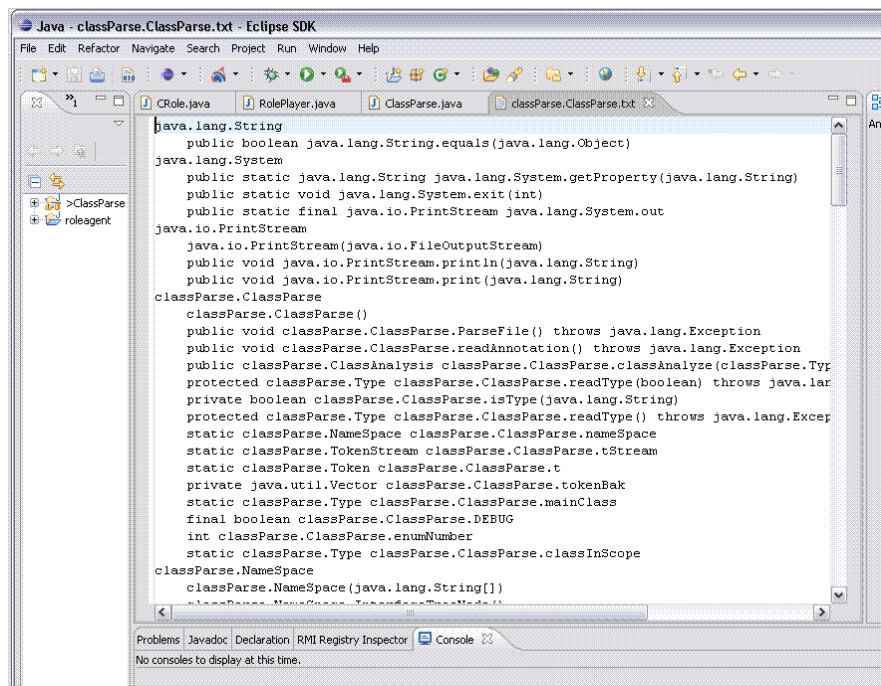


Fig. 4. Part of the Result Extracted from the class in Fig.3.

This tool has been tested by a typical Java class source file with all the required class features including inner classes and observer patterns. The result is a file including a list of class names and method patterns.

We know that a class of objects can play many roles. The above tool actually obtains a union of all the roles the objects are qualified to play. i.e., $R = \bigcup_{i=0}^{n-1} r_i$, where n is the number of roles that can be played by the class. Evidently, R is empty when the tool is applied to an abstract class or interface.

5. Comparison between Classes and Roles

Classes are tools for understanding the world. To understand a world of objects, one must classify all the objects into different classes. That is to say, all the objects can be arranged into a hierarchy of classes. Roles are also tools for the understanding of the world. To know a world of objects, one must understand the roles and the role players in the world.

Classes can be used to categorize objects; roles can as well. We can say that a class plays a specific role. When we present a class, we state the commonalities of the objects categorized by this class. When we present a role, we mean the common rights and responsibilities of the objects playing this role.

The objects described by a class are static, especially for their behaviors or services. Roles can be used to describe the dynamic behaviors and services of objects.

A class specifies the passive activities by its services. It cannot express the active activities. Roles express the active activities by its requests.

Classes are used to express the permanent behaviors for objects. Roles can be used to express the temporary behaviors for objects.

To make a software system, we need to identify all the roles in the system then write classes and instantiate instances to play these roles. Roles are really entities that separate “what to do” from “how to do”. Roles are only “what to do” and classes of objects should support “how to do”.

Classes can be divided into two categories: concrete classes and abstract classes. In concrete classes, all the methods in \mathcal{M} are implemented; but in abstract classes, not all are implemented.

Roles can be both abstract as interfaces and concrete as processes. Roles can also contain states. By specifications and aspects, we have object, interface and process roles. They are used for different purposes. They may be related with each other.

In the sense of object roles, roles are extensions of classes. Classes express static properties of objects and roles express dynamic ones. Roles and classes together express objects states and behaviors. In the sense of interface roles, classes implement roles and roles guide the design of classes. In the sense of process roles, roles are special aspects of classes and extensions of the active properties of objects.

Interface roles can be extracted from classes. Classes can be designed for playing specific interface roles.

Classes cannot complete the tasks that can be easily done by roles. For example, classes cannot tell the requests of their instances clearly, even worse, an abstract class shows nothing about requests. Classes do not express access restrictions but roles do. Classes cannot handle multiple occurrences of one object but roles can. Class hierarchies can only cope efficiently with sharing structure and behavior.

A class describes the commonalities of a group of objects statically. For example, these objects have the same structures, and same abilities. However, in fact, objects are dynamic, they can evolve. Therefore, classes lack such abilities to express objects’ evolutions. Roles can be used to express objects’ dynamic properties by having objects play roles or discard roles to show their evolutions with newly added or removed structures and abilities.

Classes have been accepted as an underlying concept and mechanism in programming but roles have not. There are no arguments regarding the properties and characteristics of

classes. Classes have concrete syntax and semantics in a programming language but roles have not.

There are still many arguments in discussing roles as programming language facilities:

1) *Is a role an instance or a class?*

From the view point of object-oriented programming, a role is certainly an object. Further more, there should be a class for a role instance. The question is still not answered. What are the states of a role instance? What are the common behaviors of a group of role instance?

2) *Do roles play roles?*

Simply, we can say, roles do not play roles and objects play roles. However, from 1), if we consider a role as an object, a role certainly plays roles. Considering the relationship between classes and objects, an acceptable trade-off is that roles normally do not play roles but roles can play roles specially controlled by language systems. It is the same situations where normally classes are not considered as objects, but classes are considered as objects by language compilers.

3) *Do roles contain states?*

From the definitions given in Section 3, Interface roles have not states but object roles and process roles do. This leads to a difficult situation for us to define a generalized role concept. If a role specifies only the services or requests without states, how could we express the evolution of objects? But if roles have states, how could we express briefly the interfaces of objects?

4) *Do roles contain behaviors?*

Process roles must specify behaviors. Even further, a process role must specify the implementation of the requests and services. If this happens, how could we keep the flexibility of the behaviors of objects?

5) *What are the relationships among roles?*

Classes clearly have inheritance, composition and use relationships. Do roles inherit other roles? Are roles composed of other roles? Do roles use other roles? These questions are not clearly answered yet. There are many challenges to deal with such relationships.

Based on the above comparison, we know that classes and roles are tools for both abstraction and classification. We can summarize the differences between of classes and roles in Table 1 and 2.

Table 1. Classes outperform roles

Properties	Classes	Roles
Instances	Yes	Yes for object roles
Service implementation	Yes	Yes object and process roles
States	Yes	Yes for object and process roles
Relationships	Inheritance, composition, and use.	Not yet clearly defined
Categories	Abstract, concrete	Object, Interface, Process
Concept	well-defined and accepted	Not yet well-defined and accepted

Table 2. Roles surpass classes

Properties	Classes	Roles
Aspects	No	Yes
Dynamic properties	No	Yes
Interfaces	Service	Both services and requests
Activities	Passive	Both active and passive
Requests	Implicit	Explicit

6. Conclusions and Future Work

Classes and roles are different concepts and mechanisms. They are used for different aspects in modeling. They cannot replace but can supplement each other. Classes have been a well-defined mechanism in programming languages for many years and there is no new development for this mechanism. Roles are still under development, they are not yet a well-defined mechanism in programming languages. Therefore, there might be new wonderful and well-accepted structures and regulations for roles to facilitate system constructions. More work should be done to investigate the well-accepted properties of roles in order to make roles become a similar mechanism to classes. With both classes and roles, our programming languages and software development methodologies will become more powerful and more easily to map the problem domain to the solution domain.

This demonstrates the requirement to introduce roles as underlying mechanisms in programming languages. It is an exciting and promising research for improving current programming languages.

Acknowledgements

This research is supported by National Science and Engineering Research Council, Canada (NSERC, No. 262075-06) and the IBM Eclipse Innovation Grant Funding. Thanks also go to Rob Alkins for his proofreading this paper and his implementation of the tool mentioned in this paper.

References

- [1] Albano, A., Bergamini, R., Ghelli, G. and Orsini, R., "An object data model with roles", in *Proc. of the International Conference on Very Large Databases*, 1993, pp. 39-52.
- [2] Ashforth, B. E., *Role Transitions in Organizational Life: An Identity-based Perspective*, Lawrence Erlbaum Associates, Inc., 2001.
- [3] Bäumer, D., Riehle, D., Siberski, W. and Wulf, M., "Role Object", *Pattern Languages of Program Design 4 Edited by Neil Harrison, Brian Foote, and Hans Rohnert*, Addison-Wesley, 2000, pp. 15-32.
- [4] Biddle, B.J. and Thomas, E.J.(Ed), *Role Theory: Concepts and Research*, John Willey & Sons, Inc., 1966.
- [5] Booch, G., *Object-Oriented Analysis and Design (2nd Ed.)*, Benjamin-Cummings, 1994
- [6] Budd, T., *An Introduction to Object-Oriented Programming (3rd Ed.)*, Addison-Wesley, 2002.
- [7] Dahchour, M., Pirotte, A., Zimányi, E., "A role model and its metaclass implementation", *Information Systems*, vol. 29, no. 3, May 2004, pp. 235-270.
- [8] Ferraiolo, D. F. and Kuhn, D. R., "Role-Based Access Control", *Proceedings of the NIST-NSA National (USA) Computer Security Conference*, 1992, pp. 554-563.
- [9] Genilloud, G. and Wegmann, A., "A Foundation for the Concept of Role in Object Modelling", in *Proc. of the 4th International Enterprise Distributed Object Computing Conference (EDOC2000)*, Makuhari, Japan, Sept. 2000, pp. 76-85.
- [10] Gottlob, G., Schrefl, M. and Röck, B., "Extending Object-Oriented Systems with Roles", *ACM Transactions on Information Systems (TOIS)*, vol. 14, no. 3, July 1996, pp. 268-296.
- [11] Gosling, J., Joy, B., Steel, G., and Bracha, G., *The Java Language Specification Third Edition*. Boston: Addison-Wesley, 2005.
- [12] Guarino, N., "Concepts, Attributes and Arbitrary Relations: Some Linguistic and Ontological Criteria for Structuring Knowledge Bases", *Data & Knowledge Engineering*, vol. 8, 1992, pp. 249-261.
- [13] Kristensen, B. B. and Østerbye, K., "Roles: Conceptual Abstraction Theory & Practical Language Issues", *Theory and Practice of Object Systems*, vol. 2, no. 3, 1996, pp. 143-160.
- [14] Murdoch, J. and McDermid, J. A., "Modeling engineering Design Process with Role Activity Diagrams", *Transactions of the Society for Design and Process Science(SDPS)*, June 2000, vol. 4, no. 2, pp.45-65.
- [15] Patterson, J. F., "Comparing the Programming Demands of Single-User and Multi-User Application", *The fourth Symposium on User Interface Software and Technology*, ACM Press, Nov. 1991, pp. 87-94.
- [16] Park, J. S., Sandhu, R. and Ahn, G. J., "Role-Based Access Control on the Web", *ACM Transactions on Information and System Security*, vol. 4, no. 1, Feb. 2001, pp. 37-71.
- [17] Pernici, B., "Objects with Roles", *ACM SIGOIS Bulletin, The Conference on Office Information Systems*, March 1990, vol. 11, no. 2-3, pp. 205-215.
- [18] Sandhu, R., Coyne, E., Feinstein, H., and Youman, C., "Role-Based Access Control Models", *IEEE Computer*, vol. 29, no. 2, 1996, pp. 38-47.
- [19] Steimann, F., "Role = Interface: A merge of concepts", *Journal of Object-Oriented Programming*, vol. 14, no. 4, 2001, pp. 23-32.
- [20] Stone, P. and Veloso, M.M., "Task Decomposition, Dynamic Role Assignment, and Low-Bandwidth Communication for Real-Time Strategic Teamwork", *Artificial Intelligence*, vol. 110, no. 2, 1999, pp. 241-273.
- [21] Zhu, H., "Role Mechanisms in Collaborative Systems", *International Journal of Production Research*, vol. 44, no. 1, Jan. 2006, 181-193.
- [22] Zhu, H., "Separating Design from Implementations: Role-Based Software Development", *Proc. of the 5th IEEE International Conference on Cognitive Informatics (ICCI'06)*, Beijing, China, July 17-19, 2006, pp. 141-148.
- [23] Zhu, H. and Zhou, M.C., *Object-Oriented Programming with C++: A Project-Based Approach*, Tsinghua University Press, 2006.
- [24] Zhu, H. and Zhou, M.C., "Role-Based Collaboration and its Kernel Mechanisms", *IEEE Trans. on Systems, Man and Cybernetics, Part C*, vol. 36, no. 4, July 2006, pp. 578-589.
- [25] Zhu, H. and Zhou, M.C., "Supporting Software Development with Roles", *IEEE Trans. on Systems, Man and Cybernetics, Part A*, vol. 36, no. 6, Nov. 2006, pp. 1110-1123.