Towards Role-Based Programming

Haibin Zhu
Department of Computer Science and Mathematics, Nipissing University, 100 College Drive, Box 5002, North Bay, ON P1B 8L7 Canada
(705)474-3461 ext 4434
haibinz@nipissingu.ca

Rob Alkins
Department of Computer Science and Mathematics, Nipissing University, 100 College Drive, Box 5002, North Bay, ON P1B 8L7 Canada
alkins.rob@gmail.com

ABSTRACT
Even though Object-Oriented Programming (OOP) is a well-accepted programming paradigm, we are required to find new approaches to improve what have not been done in OOP. Role-Based Programming (RBP) is one of such possible innovations. In this paper, we discuss the principles and activities of RBP; clarify the challenges and benefits of RBP; and describe a tool that is required in RBP. At last, we conclude the paper and point out the future work.

Categories and Subject Descriptors
D.1.5 [**Object-Oriented Programming**]; D.2.10 [**Software Engineering**]: Design -- Methodologies; D.3.2 [**Language Classifications**]: Object-oriented languages; D.3.3 [**Language Constructs and Features**].

General Terms
Design, Languages, and Theory.

Keywords
Roles, Role-Based, Programming.

1. INTRODUCTION
Object-Oriented Programming (OOP) has dominated the programming field for more than 20 years. It is no doubt a great step stone in the advance of programming. Although most programmers are still applying this style, many people are investigating new methodologies and approaches to innovate and supplement OOP. Aspect-Oriented Programming (AOP) is one of such possible innovations, which takes advantages from its kernel mechanism aspects to support separation of concerns.

Different from the mechanisms of AOP, we propose a new kind of programming, i.e., role-based programming (RBP) to support separation of concerns and related features. We can view RBP as an advance in programming methodologies. In each newly proposed methodology, the tasks of its predecessors are valid but become less important. We note the following programming methodologies:

CSCW '06, November 4-8, 2006, Banff, Alberta, Canada

- In procedural programming, composing procedures and specifying procedure calls are the most important tasks of programming;
- In OOP, composing classes, instantiating objects and specifying message passing are the most important tasks of programming;
- In AOP, composing joint points, advices and aspects is the major task of programming. It complements OOP by facilitating modularity and pulling the implementation for cross cutting concerns into a single unit [9]; and
- In role-based programming, composing roles and specifying how objects play roles are the most important tasks of programming.

Although there are some relevant researches on role-related programming [1, 2, 4, 6, 8, 10-15], there are not really available methodologies, languages, or tools to support real role-based programming.

The rest of this paper is organized as follows. We begin by reviewing past work related to programming with roles. Then, we present the principles and major activities of role-based programming. After that, we describe the challenges and benefits of role-based programming. In the following section, we describe a tool to extract role interfaces from classes. Finally, we make conclusions about our work and describe future work.

2. PAST WORK ON PROGRAMMING WITH ROLES
Pernici used the role concept to model the behavior of an object [12]. An object can play different roles at different times, and may also play more than one role at the same time. It is desired to define a class with roles as components. This method is too complex to be practical. For example, one full page fails to describe a class Car with all the roles for a car [12].

Gottlab et al. describe the characteristic features of the role concept [6]. They show how evolving objects can be handled nicely in class-based systems with roles and a role hierarchy.

VanHilst and Notkin introduce roles with both object collaboration and object evolution. In the collaboration view, a role is the part of an object that fulfills its responsibilities in collaboration [15]. Compared to classes, roles encapsulate fewer decisions and are thus more stable with respect to evaluation.

Riehle et al. state that a role model is a description of a set of object collaborations using role types. They introduce a role type to describe the view one object holds on another object. They describe a role type using an appropriate type specification mechanism [13]. A role model describes a particular aspect of an

object. Collaboration is taken as a set of roles and their relationships.

Bäumer et al. models context-specific views of an object with role-object pattern as separate role objects which are dynamically attached to and removed from the core object [1]. Their method provides a good guidance to design an object system with role concepts. It helps the role concepts be reused at the pattern level.

Patterson [11] emphasizes a role concept with the idea of an interface between objects. A user-interface is simply an elaborate mechanism for moving messages back and forth between the user object and other objects. Given the roles of users, the messages understood by them are known. The role is used to enable and disable an object's visibility and to act as a filter on the input events. Unfortunately, the work does not go far enough as to providing a real abstract structure to support more flexible role application. The follow-up work done by Steimann directly proposes and uses a formula "role = interface" in his article [14].

Kristensen [10], Depke et al. [4] and Kendall [8] summarize the common properties of roles in an object-oriented view:

1) Roles are a concept or specialization of a concept.
2) Roles can be acquired and abandoned independently.
3) Roles can be organized in hierarchies.
4) Roles model a perspective on a phenomenon.
5) Roles are bound to an existing object.
6) Roles are used for the interactions among objects.
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 objects.
10) The states and features of an object can be role-specific.

3. ROLE-BASED PROGRAMMING

From the review of past work on programming with roles, we find that there is no a systematic description of programming with roles. Many authors discuss roles at the conceptual level.

There are two limitations in the past work. One is that roles are taken as object-dependent concepts, i.e., roles are behaviors of objects. Therefore, based on the traditional view of roles, it is difficult to obtain a whole view of a role in programming. In fact, roles can be defined independently.

The other drawback of the past work is that from the viewpoint of a programmer, s/he is granted access to all other objects/classes of a language when s/he is programming. In reality, when a programmer makes an object/class, s/he only needs to access very limited number of objects/classes. When we design a class of objects, it is necessary to limit the scope of available classes. A role should provide a tool to guide the design of a class. Current languages have no such guidance.

3.1 Principles of RBP

By expanding the principles of OOP [7, 18, 19]:

- Everything in the world is an object.
- Every system is composed of objects.
- The development of a system is caused by the interactions among the objects inside/outside the system.
- Every object is an instance of a class.
- A class expresses the behavior of a group of objects.
- Classes are organized in a hierarchy of inheritance.

We have the principles of RBP:

- In a system, there are roles to be played by objects.
- Roles are integrated interfaces that confine both the requests (rights) and services (responsibilities) of objects.
- Roles are organized in request/service and super/sub relationships;
- Roles are dynamics for objects to create and evolve in a system.
- The system architecture is designed with roles.
- The system is implemented with objects filling in the architecture formed with roles.
- The system works when objects play roles.

In a society, roles, role relationships and role playing are three essential activities. To support RBP, we need to provide a new role-based programming language. In this kind of language, there must be facilitates as follows:

- Specification of roles including both services and requests. That is to say, there must be a keyword role in the language similar to class in object-oriented languages.
- Specification of the assignment of objects to roles. That is to say, there must be a simple notification to express that an object is playing a role. It is also a mechanism to specify the difference between an object with and without a role assignment.
- A mechanism to determine if an object is able to play a role. There should be a mechanism in the language preprocessor or complier to evaluate if an object is qualified to play a role. This will be an expansion of syntax checking or a grand-grain syntax checking.

Based on a RBP language, we have the phases or activities of programming as follows [16, 21]:

1) Architecture design;
2) Object implementation; and
3) System integration.

In the architecture design, role specification and role relationships are the major tasks; in the object implementation, class design and implementation are the major tasks; in system integration, objects and roles are combined together to make a whole system executable.

In these three phases, 1) and 2) can be done in parallel. This makes it possible for shortening the programming time. For more details of the whole life cycle of role-based development, please refer to [16, 21].

3.2 Architecture Design

People are mainly concerned with specifying roles and the relationships among roles. In design, roles are created and put into a role pool arranged with role relationships. These relationships include request/service, super/sub (promotion), and conflict. Where, the request/service relationship expresses that the object playing one role may request the object playing another role; the promotion relationship expresses that the object playing a sub role may play the super role when some conditions are met; the role conflict relationship expresses that an object cannot play the conflict roles at the same time. We may call the people mainly doing architecture design as designers. The product of this phase is a role structure.

3.3 Object Implementation

In phase, people are mainly concerned with implementing objects/classes that can play roles. The tasks of object implementation are to perform the services based on the provided requests with special objects/classes. We may call the people mainly doing object implementation as programmers. The product of this phase is objects/classes.

In RBP, programmers may work in parallel. Suppose there are M objects, N programmers making objects/classes and averagely each programmer completes one object/class at T time units. The time to complete M objects/classes is MT/N. Therefore, adding more (Δ) programmers will certainly shorten the time to complete the task in MT/(N+ Δ). This potentially breaks the famous Brooks' law, i.e., increasing programmers does not speed up the software development [3].

3.4 System Integration

With roles and objects, a software product can be obtained by integration, i.e., assigning objects with roles, or having objects play roles. This step will build the deliverable software product. At this step, objects are assigned to match a role or roles to play. If each role has a relevant object to play and the object can provide the space and speed requirement of that role, the system is complete.

To complete this task, we need specialists to couple objects and roles. They know the objects and the basic specification of roles. They are match-makers between roles and objects. They are the bridges between design and implementation. Match-making is their primary task. These specialists are totally new type of team member when compared with the team members in a traditional software development team. To facilitate this match-making, we need two tools: one tool is to extract the interface of a traditional Java class (not only the service interface but also the request interface), and the other helps check if an object is able to play a role based on the extracted interface. We can call the people mainly doing system integration as integrators. The product of this phase is a deliverable system.

Evidently, the above three steps can be done by specialists such as designers, programmers, and system integrators with different experiences and trainings. The integration step shows a lot of scaling capabilities based on how many objects play a single role. The efficiencies of different objects provided by different programmers may be different. Based on these differences, managers of the development team have a concrete evaluation criterion for programmers.

Above all, the skills of designers, programmers, and integrators are totally different. Designers are experts of role specifications, role relationship analysis and role structure design. Programmers are experts with a special language to create objects to be qualified to play roles specified by designers. Integrators are specialists who match objects with roles. They have different considerations, expertise, and knowledge. This specialization really separates designers, programmers and integrators.

4. CHALLENGES AND BENEFITS

4.1 Challenges

To make RBP a reality, the top challenge is to determine at what level of abstraction a role should be located. Is a role specified as a class? Is a role specified as an object? Is a role specified as a new entity that is neither a class nor an object? All these questions should be clarified before designing a RBP language. The past work only considers roles as concepts in modeling and makes roles a guidance of programming.

To support RBP and follow the principles of RBP, the next challenge is the language structures. We need a language structure to specify a role. After specifying a role, we need to provide a mechanism or a language structure to express if an object has played a role and if an object is playing a role. We also need a tool to determine if an object is qualified to play a role and if an object is appropriate to play a role.

To implement the mechanisms of RBP, the concrete challenge is the interface specification, i.e., how to specify the interfaces (requests and services) of a role. It seems solved in traditional programming languages. However, this kind of solution is at the syntax or grammar level. The most difficult one is located at the semantics level. That is to say, to express an interface exactly not only at the syntax level but also at the semantics level is the most difficult task for RBP. This challenge is similar to that in service science and service architecture, such as, service discovery and service retrieval.

4.2 Benefits

With RBP, we could anticipate the following benefits:

1) Specialization. Specialization is an efficient way for mass production. It is also very important in software development and has many benefits [16]. The development of programming methodologies can be viewed as gradually improving the support of specialization and helping "divide and conquer". Structural programming requires programmers have special skills for making specific procedures; object-oriented programming requires programmers have special skills for making specific classes; aspect-oriented programming requires programmers have special skills for making specific aspects. With RBP, people are specialized in different levels, role specification and relationships, object design and implementations, and system integrations by matching objects and roles.

2) Separation of concerns. With RBP, we will have special experts who specialize in role structure design; in role-object matching and in making objects/classes. They are all educated and trained with different skills and special knowledge.

3) Better productivity. Programmers concentrate on object/class design and implementation. They do not care too much about the whole system design or communications with other team members. They instead use their time to understand the role requirement and do their best to make objects/classes meet their requirements.

4) More reusable components. With RBP, we can evaluate a programmer's workload or performance by how many objects/classes s/he built are used in a system. With the role requirement in system architecture, programmers will try their best to make their objects/classes more reusable.

5) More maintainability. With RBP, the syntax errors are mainly concerned with the matching between roles and objects/classes. The language tools will find the interface matching errors between objects/classes and roles. If some

roles are not properly or efficiently played, we could quickly find another object/class to replace this object/class.

5. THE INTERFACE EXTRACTION TOOL

To complete the first step towards RBP, we start our work from a solid base, i.e., roles are defined as an integrated interface shown in Fig. 1. This tool 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 role defined by both responsibilities and rights [16, 17, 19]. Because it is easy to extract the service interface of a class, we concentrate on the request interface in this section.

The tool is actually a simplified Java compiler. It analyzes a class and reports the types that the class uses as well as any constructors, methods and fields of those types. In order to accomplish this task, the parsing utility closely mimics the early actions of a compiler. The Java source file syntax is parsed according to the Java grammar rules [5] and all types are fully resolved.

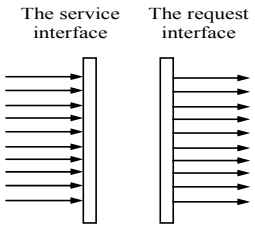

Fig. 1 A role is a wrapper of an object

The parsing utility relies on one very important assumption, that is that the class to be parsed is a legal Java program. Instead of checking for errors, it relies on the rules of Java grammar [5] to simplify the formation of an accurate parse of the file. The assumption that the file is a legal program can be enforced by invoking the Java compiler prior to parsing the source file. In fact, this is a necessary step since the program uses the Java reflection classes to open the class file of the program being parsed, and query it for fields methods, and inner classes. The class must be queried because Java allows for forward references.

The fist stage in parsing is to devide the file into a stream of tokens. Unicode escape sequences are treated at this point. The program then groups the characters from the file to be parsed into tokens, each token is identified as an operator, separator, identifier, keyword, or literal. Literals are further sub-categorized into their types such as string, boolean, character, etc. This stream of tokens is passed on to a class parse. The class parse takes the incoming stream of tokens, and constructs an abstract analysis of the class. Import and package statements are passed on to the name space where they will later be used to resolve type and variable names.

The class definition and the type parameters declared in the class are passed on to the name space. Within a class definition the parsing utility will recognize all method, field and static initializer declarations, as well as the declarations of a nested or inner classes. Any methods encountered initiate a method parse as outlined below, also fields declared with an initializer initiate a method parse since they may contain the declaration of an anonymous class. Any method type parameters or argument type and names encountered are passed on to the name space.

All types located within any part of the tool being parsed are immediately passed to the name space where they are resolved to their fully qualified name. For each type, the name space verifies if the appropriate class file exists. Within the name space is a collection of fully qualified class names from the package and import statements including the implicit java.lang.* import. Any import statements ending in a wild card token initiate a search of the file system (in the working directory as well as any directories supplied as arguments to the program) for all top-level classes in that package. Inner and nested class names are stored with a mapping from their abbreviated name to their fully qualified name supplied by the class parse, these abbreviated names are added and removed as the class parse proceeds through the different outer or inner classes and the abbreviated names go in and out of scope.

Scope is accommodated for by adding a depth value to each type parameter, inner class name or variable name and a global counter that is incremented or decremented by the class or method parse. Each time the depth is decreased any scope-sensitive items with a depth greater than that of the global depth counter are released. When resolving a name, the item with the greatest depth is used.

Recursion is essential for a language grammar. As a result, the methods used to parse a source file are designed for recursive calling of each other. Each identifier in a type can contain type arguments, which are in fact, types. Thus each of those types can declare type arguments and so on. Classes can contain classes, or methods which contain classes. The method parse is designed around the highly recursive rules for Java grammar as outlined in the Java Language Specification [5]. As a result it is a collection of short methods which call each other repeatedly.

The method parse makes no attempt to build a parse tree from the source file, it is only interested in the fields accessed and methods called within a given method. Any declarations encountered within a method are passed along to the name space. Any control statements such as for, do and while loops, if statements, switch statements, etc. are read but essentially skipped. Any non-primitive declarations are reported. Any classes encountered within a method recursively initiate a class parse. All expressions parsed return a type.

It is within an expression that method invocation and field access take place. All expressions begin with a primary, all primaries have an associated type. The rules for Java expressions are highly recursive. For example, a primary may be an expression in parenthesis, if so, the type of the primary is the type returned by parsing the expression within the parenthesis. Any comparison operators connecting parts of expressions will result in type boolean. Algebraic or bitwise infix operators will widen the type of either expression fragment accordingly.

If the name of a variable is encountered it is passed to the name space, which will return a type. Static references are also passed to the name space to get their fully qualified type. Literals encountered will already be marked with a type. Any fields accessed will be located by using the Java reflection classes, have their type determined then will be reported. This approach makes it possible to determine the type of a methods arguments and the class in which to start searching for a method, should one be encountered.

Once a method invocation is recognized, the type of its class is known, and a set of argument types have been determined, locating the method is not a simple task. Simply querying the class is not sufficient, the compiler allows for many type conversions to take place or the method may be defined in the superclass of the determined type. A complex algorithm is used to load all methods of that class and preform boxing/unboxing conversion, widening primitive conversion or widening reference conversion and to search each superclass and all interfaces extended by those superclass's.

Determining a methods return type is no-trivial as well, a method may return a type that is a type parameter for the method, and if that method is called without type parameters it is necessary to infer the type of that parameter by the types of the arguments. Once the return type is determined, it is passes upwards and the parse continues.

As the above types, methods and fields are being reported, duplicates are removed and the information stored. When the end of the top-level class is reached, the information is saved in a text file with the name of the fully qualified class in the working directory. Within the file is a list of all types used and each method, constructor or field of that type is stored below it indented by one tab.

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.

6. CONCLUSIONS AND FUTRUE WORK

The introduction of roles will fundamentally change programming style. RBP has many emergent benefits to programming. It may change the software team management, software design and programming [16, 20].

RBP may bring in exciting advancement for programming. Role-based approaches have been noticed in different applications such as intelligent systems, system management, human-computer interactions and service sciences. It is valuable to investigate thoroughly role-based programming and role-based methodology with reasonable efforts.

The emergent future jobs include:

1) To standardize the specification of roles so as to make the architecture design with roles become reality and make the interface extraction tool really available in the system integration phase.

2) To design a role preprocessor for Java. The preprocessor will transfer a program made by the proposed role mechanisms to a Java program.

3) To design a totally brand new language. This language will facilitate role-based programming including the whole life cycle of role-based software development, such as team management, analysis, design, constructions and maintenance.

7. ACKNOWLEDGMENTS

This work was supported in part by an Eclipse Innovation Award from IBM and by the National Science and Engineering Research Council, Canada (NSERC, No. 263075-06).

8. REFERENCES

[1] Bäumer, D., Riehle, D., Siberski, W. and Wulf, M., "Role Object", In *Neil Harrison, Brian Foote, and Hans Rohnert, Pattern Languages of Program Design*, Addison-Wesley, 2000, 15-32.

[2] Becht, M., Gurzki, T., Klarmann, J. and Muscholl M., "ROPE: Role Oriented Programming Environment for Multiagent Systems", *Fourth IECIS International Conference on Cooperative Information Systems*, (Sept. 1999), 325- 333.

[3] Brooks, F.P., Jr., *The Mythical Man-Month*, Addison-Wesley, 1995.

[4] Depke, R., Heckel, R. and Kuster, J. M., "Roles in Agent-Oriented Modeling", International Journal of Software Engineering and Knowledge Engineering, 11, 3 (2001), 281-302.

[5] Gosling, J., Joy, B., Steel, G., and Bracha, G., *The Java Language Specification Third Edition.* Boston: Addison-Wesley, 2005.

[6] Gottlob, G., Schrefl, M. and Röck, B., "Extending Object-Oriented Systems with Roles", *ACM Trans. on Information Systems*, 14, 3 (July 1996), 268-296.

[7] Kay, A., "The early history of Smalltalk", *The Second ACM SIGPLAN conference on History of programming languages (Cambridge, Massachusetts, USA, 1993)*, 69-95.

[8] Kendall, E. A., "Role Model Designs and Implementations with Aspect Oriented Programming", *Proceedings of ACM 1999 Conference on Object-Oriented Programming, Systems, Languages and Applications (OOPSLA'99) (*CO, USA, Nov. 1999), 353-369.

[9] Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C., Loingtier, J.M., and Irwin, J., "Aspect-Oriented Programming", In *Proc. of the European Conference on Object-Oriented Programming (ECOOP)(Finland. Springer-Verlag LNCS 1241. June 1997).*

[10] Kristensen, B. B. and Østerbye, K., "Roles: Conceptual Abstraction Theory & Practical Language Issues", *Object-Oriented Systems*, 2, 3 (1996), 143-160.

[11] Patterson, J. F., "Comparing the Programming Demands of Single-User and Multi-User Application", In *Proc. of the 4th Symposium on User Interface Software and Technology*, ACM Press, Nov. 1991, 87-94.

[12] Pernici, B., "Objects with Roles", *ACM SIGOIS Bulletin,* 11, 2-3 (March 1990), 205-215.

[13] Riehle, D., Brudermann, R., Gross, T. and Mätzel, K. U., "Pattern Density and Role Modeling of an Object Transport Service", *ACM Computing Surveys (CSUR)*, 32, 1 (March 2000), 1-6.

[14] Steimann, F., "Role = Interface: A merge of concepts", *Journal of Object-Oriented Programming, 14, 4* (2001) 23-32.

[15] VanHilst, M. and Notkin, D., "Using Role Components to Implement Collaboration-Based Designs", In *Proceedings of ACM 1996 Conference on Object-Oriented Programming, Systems, Languages and Applications (OOPSLA'96)*, CA, USA, 1996, pp. 359-369.

[16] Zhu, H., "Separating Design from Implementations: Role-Based Software Development", In *Proc. of the 5th IEEE International Conference on Cognitive Informatics (ICCI'06) (Beijing, China, July 17-19, 2006*), 141-148.

[17] Zhu, H. "Role Mechanisms in Collaborative Systems", I*nternational Journal of Production Research*, *41, 1* (Jan. 2006), 181-193.

[18] Zhu, H. and Zhou, M.C., "Methodology First and Language Second: A Way to Teach Object-Oriented Programming", In *Companion of OOPSLA'03 (California, USA, Oct., 2003)*, 140-147.

[19] Zhu, H. and Zhou, M.C., *Object-Oriented Programming with C++: A Project-Based Approach*, Tsinghua University Press, Beijing, China, 2006.

[20] Zhu, H. and Zhou, M.C., "Role-Based Collaboration and its Kernel Mechanisms", *IEEE Trans. on Systems, Man and Cybernetics, Part C, 36, 4* (July 2006), 578-589.

[21] Zhu, H. and Zhou, M.C., "Supporting Software Development with Roles", *IEEE Trans. on Systems, Man and Cybernetics, Part A, to appear* (2006).