

# From OOP to SOP: What Improved?

Haibin Zhu, *Senior Member, IEEE*

**Abstract**—Service-oriented programming (SOP) is abstracted from industrial application development practices and attracts more and more concerns from programmers and developers. The major question programmers and developers may ask is “is SOP a sound direction for software development?” or “should I learn and apply this paradigm in my work?” This paper clarifies this question by tracing programming style development and comparing the fundamental principles, key concepts and key components of SOP and object-oriented programming (OOP), discusses how SOP can meet more requirements for software development, demonstrates how SOP could promote component-based development. By examining the basic elements of SOP, this paper points out the potential problems that still require to be solved. This paper concludes that SOP has demonstrated many innovative aspects as compared with OOP and there are still some problems to solve. This paper predicts that SOP is a sound direction to improve programmers’ productivity and it is worthwhile putting more efforts on this kind of research and practice.

**Index Terms**—Service, Object, OOP, SOP

## I. INTRODUCTION

Service-oriented programming (SOP) is proposed to support reusing and enhancing distributed system development (Bieber, 2001, Sobolewski, 2002, He, 2003, Kolonay, 2004,). SOP originates from the concepts in object-oriented and component-based development. It also expands these with distributed computing mechanisms (Momentum, 2003).

Object-oriented programming (OOP) has been proposed for more than thirty years. One of the pioneers of object-oriented programming, Alan Kay, obtained the 2003 Turing Award. This demonstrates that object-oriented programming has undoubtedly made great contributions to the computing community and information industry. However, object-oriented programming as a paradigm is highly abstract. Even though object-oriented programming languages provide syntax restrictions to programmers, this programming paradigm leaves too much practical flexibility for programmers to work according to their own familiar styles.

The author experienced teaching many students in object-oriented programming courses and encountered many instances of non object-oriented programming with object-oriented programming languages, especially with C++ (Zhu,

2003, Zhu 2004a). From the point of view of engineering, specifications that are too abstract without rigorous restrictions may lead to various unpractical productions. Therefore, more concrete disciplines are required to improve OOP.

Separation of concerns is welcome in programming and modeling fields, such as aspect-oriented programming (Kiczales, 1996) and role-based modeling (Zhu, 2004b). That SOP supports separation of concerns is also an asset for application development (Li, 2003).

This paper is arranged as follows. Section 2 briefly reviews the programming development history and the evolution of programming styles; section 3 compares the fundamental principles of SOP and OOP; section 4 illustrates SOP and OOP mainly on the aspects of key concepts and key elements; sections 5 demonstrates the improvements of SOP to component-based development; and at last, section 6 concludes that SOP is a sound direction at the viewpoint of software development.

## II. THE DEVELOPMENT OF PROGRAM PARADIGMS

The objective of programming is to make a good program. What do we mean by a good program? A good program is considered normally by its correctness, efficiency, readability, and user-friendly interface (Zhu, 2005).

- It should run correctly to solve its intended problem;
- It should run efficiently to avoid wasting time and memory space;
- It should be readable to allow other programmers to understand, modify and improve it easily;
- It should have a user-friendly interface that is intuitively easy to learn and use.

In the past, the evolution of programming techniques has always been reflected in programming languages first. Therefore, in a common sense, developing programming techniques meant to develop languages. These languages should be more expressive and could be more easily used to construct complex software systems. The eventual aim was to help programmers make good programs in an efficient way.

With the development of applications, programming is becoming a wider concept that covers many aspects of software engineering. Sometimes, programming is a synonym of software development. The following formula is well accepted in developing a large program (Derema and Korn, 1976):

Programming in the large =

- Programming in the large +
- Programming in the small.

Haibin Zhu is with the Department of Computer Science and Mathematics, Nipissing University, 100 College Drive, North Bay, Ontario P1B 8L7, Canada (phone: 1-705-474-3461 ext 4434; fax: 1-705-474-1947; email: haibinz@nipissingu.ca).

This means that programming in the large divides the tasks into sub-tasks small enough to be solved by programming in the small. Definitely, SOP views programming as programming in the large.

As another well-accepted formula for programs, a program = data structures + algorithms. People work hard to improve programming with different concentrations on data structures, algorithms or both. Procedural or structured programming concentrates on algorithms, i.e., how to process. OOP concentrates on data structures, i.e., objects are considered at first and algorithms are subordinate to data structures. SOP concentrates again on algorithms (how to process), but at a higher level of consideration. As is stated on the website of Momentum Software Ltd:

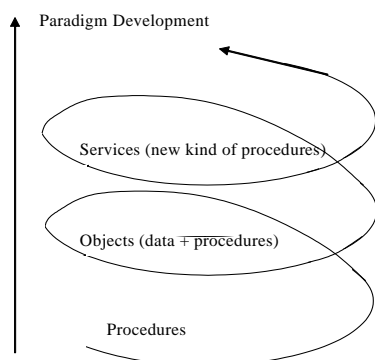


Fig. 1 The development of programming paradigm

“Software development has gone through a number of paradigm changes. We have moved from using a structured programming technique to object-oriented programming, then to component-based programming and now we are moving to a service-oriented programming style (Momentum, 2003).”

Fig. 1 shows the spiral development of programming styles. SOP is actually the next generation of procedural (structured) programming beyond object-oriented programming.

In fact, in all the mathematical models for complex systems, people try to simulate the major processes in the systems. That was why the pioneers of programming tried to make programs with procedures. A procedure is the first step of abstraction of process simulation. When people mention software engineering, they are mainly concerned with the processes of developing software. Processes are taken as the key fundamental concepts for software engineering in almost all software engineering textbooks (Pressman, 2004, Kan, 2003, Braude 2001). That is to say, “how we should process” is the major concern of software engineering. Therefore, programming and software engineering are all concerned about processing. From the viewpoint of programming, structured programming deals with processes by procedures and controlling structures such as sequential statements, branch statements, and circulate statements; object-oriented programming deals with processes by objects, classes, classification, inheritances and polymorphisms; service-oriented programming deals with processes by more advanced elements such as contracts, connectors, services, containers

and contexts (Bieber, 2001).

To produce good software, there are many trade-offs for software developers to consider. Flexibility is one of them. We practitioners all know that more flexibility means less efficiency. That is why the first pure object-oriented programming language Smalltalk has not been used widely in industry. With the compatibility of C and high efficiency, C++ prominently occupied the OOP language market for a long time before Java was accepted as a new competitor with its networking, security and web application support. Even now, C++ is still firmly supported by its followers from many areas requiring efficiency and real-time processing.

SOP makes a new balance between flexibility and efficiency. Different services can be selected by the server consumers based on their requirements on efficiency or flexibility. These selections will be reflected in the applications provided by the service consumers.

As for productivity, object-oriented methodology has not reached its goal in software development with the promise of high productivity (Potok, 1999). This situation is due to its high level of abstraction. We need more concrete guidelines for programmers to make program development easier to control and maintain.

SOP improves productivity by providing easily reusable components, i.e., services (see section 5).

Considering the software development life cycle (SDLC), OOP introduces managerial thoughts into programming because there are many management requirements in developing large programs. In management, there is a concern of management unit grains, i.e., the size of the unit. However, because everything is an object in OOP, the advantages of management in object-oriented programming become less competitive because a programmer would like to manage their own program components in their own way and at any level of grains. In other words, there are no restricted management disciplines in OOP. SOP incorporates services as larger grains of program components to help manage the development more efficiently.

Loose coupling is a good rule in OOP. SOP supports loose coupling among interacting software components with two architectural constraints (He, 2003):

- Simple and general interfaces to all the involved software components. The interfaces state only generic semantics. They should be universally available for all providers and consumers.
- Descriptive messages. System behaviors are prescribed by messages determined by the contracts. A contract allows new versions of services to be introduced without bothering old services.

From the perspective of inheritance, we can understand the advantages of SOP compared with OOP. OOP is a superclass of SOP and SOP will do more concrete jobs than OOP.

Comparing SOP with OOP, OOP concentrates on creating objects that contain both states and operations (sometimes called services). SOP stands on OOP. It implements services

by using OOP techniques. These services themselves provide more reusability of business logic and can be used in various applications. “OOP focuses on what objects an application consists of, while a SOP approach focuses on the application’s functionality, or in other words, what the application does (Ireland, 2002).”

The advantages of SOP compared with OOP can be listed as separation of concerns, large grains of processing units, loose coupling, and descriptive messages.

### III. FUNDAMENTAL PRINCIPLES

Even though there had been many arguments in object-oriented programming for many years, it is well-accepted that object-oriented programming possesses the following general principles (Zhu, 1998, Zhu, 2004b, Zhu 2005):

- Everything in the world is an object (Kay, 1993);
- Every system is composed of objects, and certainly a system is also an object;
- The evolution and development of a system is caused by the interactions among the objects inside or outside the system;
- A message is a way to activate a method of an object;
- The interactions among objects are expressed by sending messages that are requests to invoke objects’ actions;
- Each object is an instance of a class which shows the commonality of a group of objects; and
- Each class might inherit another class which is called a superclass while it is called a subclass.

However, there are so many variants in applying these principles that there are not the same programs developed for the same requirement. Smalltalk, C++, and Java programmers have experienced different practices due to these languages’ different terminologies and constructs even though they all are called object-oriented programming languages. The situations for programmers to provide different solutions based on the same requirement are still unchanged after many years of practice in OOP.

OOP has pointed out the highest level principles. SOP principles are specialization of the OOP principles. SOP improves OOP by providing more concrete and operable guidelines for program developers. Therefore, the fundamental principles of SOP can be listed as follows:

- Every system is composed of services and service requests;
- Services are implemented and provided by service providers;
- Services are managed by service registers;
- Service consumers could issue requests to proxies of service providers;
- Proxies find and build connections with service providers.
- Requests should bring all the data to be processed by the services; and

- Service consumers finally provide users or clients with services.
- “Services are used to divide larger applications into smaller discrete modules (Momentum, 2003).”
- “Services are integrated via service composition mechanisms to create larger applications (Momentum, 2003).”

With these principles, SOP provides an innovative way to organize applications so as to make application development paradigms progress.

### IV. THE KEY CONCEPTS AND ELEMENTS

We could analyze the evolution of the key concepts of the different programming paradigms to understand the innovations of SOP.

*Unstructured Programming: Statements or instructions.* A statement or an instruction is a signal or a group of signals that command the equipment to function.

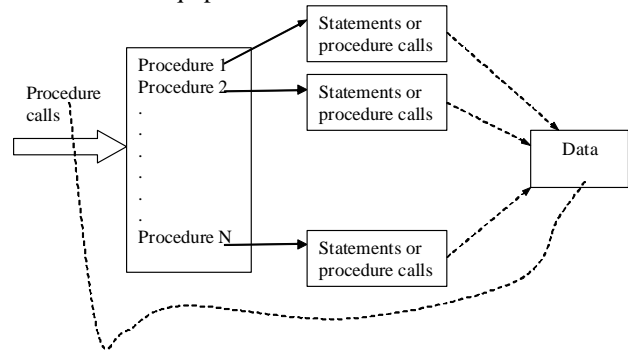


Fig. 2 Procedures

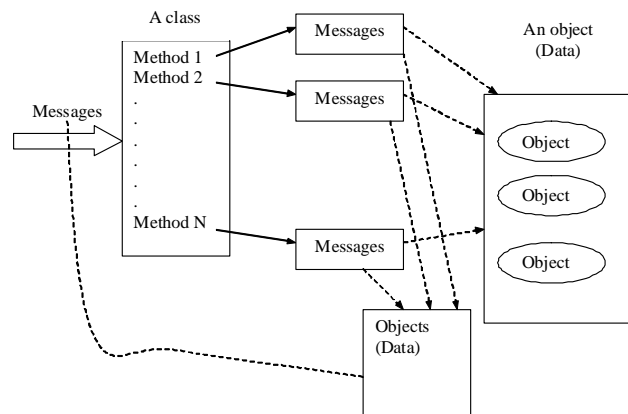


Fig. 3 Methods and messages

*Procedural (Structured) Programming: Procedures* (Fig. 2). A procedure is composed of statements that do one or more computations, the statements can also be the calls of other procedures. At this time, the computations are restricted by mathematical computations. Even though a procedure call can take large data as parameters, the parameters with large quantity of data are generally dealt with references. The key idea of procedural programming is that procedures are first considered and data is secondary to the procedures. Generally,

procedures do not own the data on which they are operating. In the figures 2, 3, and 4, we use arrows to express implementations, dashed arrows to express accessing operations and dashed lines to express references.

*Object-Oriented Programming: Methods (Fig. 3).* A method actually describes how to accomplish a process of a task. Its promotion from a procedure is demonstrated because it can create new objects and send messages to these objects to accomplish tasks. Similarly, even though a message can take complex objects as arguments to activate a method, our major design concern is dealing with the objects that accept the message. In OOP, an object (accepting messages) is considered first and the messages and the parameter objects are secondary to the object.

*Service-Oriented Programming: Services (Fig. 4).* A service is a contractually defined behavior that can be implemented and provided by a component. It can be used by another component based on the contract (The Openwings, 2002, Stevens, 2002). The key promotion from methods is that it uses more complicated objects to accomplish the tasks relevant to the service. SOP seems like procedural programming in that the request to a service is similar to a procedure call. In fact, procedural programming concentrates on how to process formatted data and this format is specified by the procedure. But SOP takes the data to be processed by the service at first, i.e., customer first. A service should be able to process large data in different formats confined by a contract. The data format is specified by a contract but not by the service. This change facilitates the development of distributed systems with the separation of the original data and the processed data. This separation greatly improves the security of the applications developed by services. Remote Procedure Calling (RPC) tried to support distributed programming with the style of procedural programming, but it could not support large data as parameters (Coulouris, 2001).

Besides, in procedural programming large data is processed only by call-by-reference and in SOP large data is processed by transferring messages. We do not use the term of “call-by-value” because traditionally, it only processes simple values. OOP is inherently distributed from principles, but it could not support distributed programming without the help of special distributed mechanisms such as Common Object Request Broker Architecture (CORBA) and Remote Method Invocation (RMI). Here, SOP’s services support distributed computing by selecting owning the data to be processed or owning the access rights to the data based on the data quantity and the accessibilities of data.

From the four key concepts in different paradigms, we find that processing is the major concern and their differences are on how to organize the processing. The grain of a processing unit (such as a procedure, a method or a service) is becoming larger and larger. The larger the processing grain, the easier to incorporate management. Therefore, for large program development, SOP is a sound direction.

Besides basic concepts, we can also analyze the basic elements of SOP to find its innovations compared with OOP.

- SOP’s contract is a specification of the way a service consumer interacts with the service providers (The Openwings, 2002). It confines the syntax and semantics of a service. Compared with the message patterns of OOP (applied in Smalltalk), it explicitly defines the request syntax and semantics. It improves the message patterns of Smalltalk or the function specifications in C++ and Java in that it incorporates preconditions and post conditions.

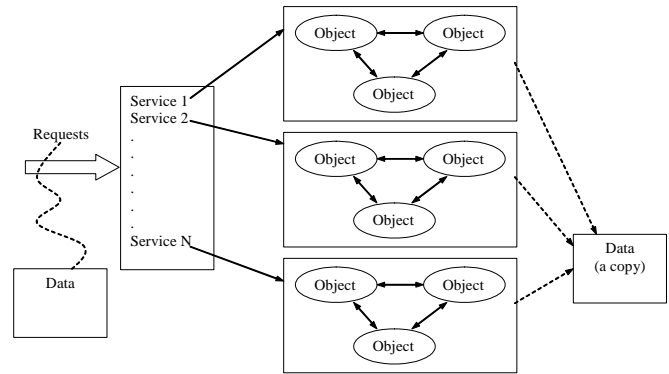


Fig. 4 Services

- SOP’s service is a reusable computing element. It is independent of platforms, protocols, and deployment environments. These services will not be restricted by programming languages. Compared with OOP’s classes as reusable components, they could only be used within one OOP language such as Smalltalk, C++, or Java.
- SOP’s connector is an encapsulation of transport-specific details for a specified contract. It is an individually deployable element (Bieber, 2001). This element makes SOP very powerful to develop distributed systems. In OOP, there is no such concrete facility to accommodate distributed computing, even though objects conceptually are distributed.
- SOP’s container is an environment for executing services that manage availability and code security (Bieber, 2001). There is no such facility to support this requirement imposed by distributed computing.
- SOP’s context is an environment for deploying plug and play services. It imposes the details of installation, security, discovery, and lookup (Bieber, 2001). There is no such facility in OOP that specially support the requirement of developing distributed applications.

## V. SOP AND COMPONENT-BASED DEVELOPMENT

Reusing has been an ideal for all programmers for more than thirty years. Component based development is one of the possible solutions to this requirement. The initial idea of component-based development is learning from the hardware industry to reuse hardware components such as integrated circuits (IC), large-scale integrated circuit (LSI), very large scale integrated circuits (VLSI), chips, chip-set, board, etc (Zhu, 1989). However, component-based development has not accomplished as much as expected a decade ago. The situation

is not satisfied hitherto because of the following reasons (Budd, 2001):

- Producing reusable components is difficult and the benefits cannot usually be realized within a short period of time and a single project. There is usually little incentive for programmers to strive toward reusability.
- Each new problem is a typically different set of behaviors. It is often difficult to design a truly useful and general-purpose software component the first time.
- Many programmers and managers tend to rely on their own development team and lack the confidence of reusable components developed by other teams.
- Many programmers have little formal training or have not kept pace with recent programming innovations; they may not be aware of the mechanisms available for the development of reusable software components.

In fact, reusing hardware components is well-accepted, because in hardware engineering understanding a component is much easier than making it. However, in software industry, a software component is either too simple or with too complicated specifications. If it is too simple, the programmers believe that they can do by themselves. If the specifications are too complex, the situation is that after understanding the component's specification the programmers believe that they can make a better one by themselves. Therefore, traditional software components could not be designed and applied in the same way hardware components are. The key is to make specifications much simpler than the implementation logic.

A service as a reusable unit is more appropriate to reusing software, because the logic of a service is generally so complicated that it is much easier for the service consumers to understand the contract than to implement the service by themselves. In this way, service consumers do not care and cannot care about the details of service implementations. This is a large step towards reusable components. With OOP, whichever language is used, the separation of implementation and specification are generally located at the conceptual level but not at the practical level. That is to say, many component users would like to know the details of the components and try to modify or "improve" the components.

SOP supports reusing with contracts and services. The "register, find, bind and execute" paradigm is common in our daily life and it is successful in providing normal services. By empirical reasoning, we can also find a good application in distributed program development.

With a service registry that is a directory on a distributed or networked environment, we can manage service contracts dynamically. After finding a service, asking for the service is nothing but just providing the data the service will process. This style completely blocks the intention of intervening with the details of the service implementation.

Even though SOP introduces contracts as the interface between a service provider and a consumer, there are still

problems as how to make the contract easily understood. We should not create a new position of attorney who is professional to help service consumers sign a contract to use a service.

Generally, a contract is defined as an abstract agreement that is used at both compile-time and runtime. It contains functional aspects and quality of services (QoS). The questions are:

- How to specify the contracts in an easy form?
- How to negotiate the QoS?
- How to make sure the contract has been accomplished?

A service proxy finds a contract and a reference to the service provider in a registry and formats the request messages and executes the request on behalf of the service consumer. The problems are who provides the service proxy? Is it the consumer or provider?

We cannot do everything by ourselves and we need to clarify our own roles. Asking experts to do something that is difficult for us to do is efficient and common. Consuming a service is usually less expensive and more effective (He, 2003). We call it "separation of concerns", and it is regarded as a good principle of software engineering (Kiczales, 1996).

All the above questions still require further research. From the viewpoint of separation of concerns, role-based systems aimed to separate a class into different concerns to describe the collaboration among objects (Zhu, 2004b). Compared with the terminologies of SOP, responsibilities are services and rights are requests. Therefore, role mechanisms could be integrated into providing better services by the register, publish, find and request paradigm. Therefore, role mechanisms are a possible way to simplify and make it easier to understand, register, find and apply services based on their role specification, role transition and role facilitation.

## VI. CONCLUSION

To conclude this paper, we can state that SOP is a sound direction to improve application development, especially for distributed applications based on the web.

We can summarize the improvements of SOP compared with OOP on the different aspects of requirements for good programs as follows.

- Readability: OOP applies some programming tricks such as ".h" files and ".cpp" files to separate specifications with implementations in C++ and message patterns in Smalltalk. SOP' contract provides more facilities for readability, i.e., the service consumers only need to understand the contracts.
- Efficiency: OOP's efficiency is lowered than structured programming in general. SOP could improve the efficiency of a component by the service provider with special consideration.
- Maintainability: polymorphism and subclassing are normally used to support maintainability for OOP, but there are still some problems to change a class that have subclasses. The application may concentrate on

their business logic because they do not care and they can not care about the service providers.

- Flexibility: subclassing is the major method to support flexibility of OOP, overloading, overriding and polymorphisms are concepts to support flexibilities of subclassing. However, many programmers do not like too much polymorphism because of its low readability and low efficiency. By SOP, service consumers could ask for different services even though their functions are similar in order to meet the special requirements.
- Productivity: a too high-level abstraction leads to too much flexibility in program development. The result is that there is not a significant productivity enhancement by OOP. In SOP, with services as easily reusable components in applications, application development is definitely improved.

SOP is a good way to integrate different web-based systems. It can be developed into a fundamental methodology or architecture including a group of mechanisms such as component development, service development and contract specification.

There are still some problems that need more re-search such as contract specification, service registration mechanisms, service proxy mechanisms and QoS negotiation mechanisms.

#### ACKNOWLEDGMENT

This research is partly supported by the IBM Eclipse Innovation Grant Funding. Thanks to Pierre Seguin for his proofreading this article.

#### REFERENCES

- [1] Bieber, G. and Carpenter, J., 2004. Introduction to Service-Oriented Programming (Rev 2.1), <http://www.openwings.org/download/specs/ServiceOrientedIntroduction.pdf>, 2001
- [2] Budd, T., 2002. An Introduction to Object-Oriented Programming (3<sup>rd</sup> Ed.), Addison-Wesley, 2002
- [3] Controneo, D., Di Flora, C. and Russo, S., 2003. Improving dependability of service oriented architectures for pervasive computing, Proc. of the Eighth International Workshop on Object-Oriented Real-Time Dependable Systems (WORDS 2003). Jan. 15-17, 2003, pp. 74- 81
- [4] Coulouris, G., Dollimore, J., Kindberg, T., 2001. *Distributed Systems: Concepts and Design* (3<sup>rd</sup> Ed.), Addison Wesley, 2001
- [5] DeRemer, F. and Kron, H., 1976. Programming in the Large Versus Programming in the Small, IEEE Transaction on Software Engineering, vol. 2, no. 2, June 1976, pp. 80-86
- [6] Kiczales, G., et al., 1996. Aspect-Oriented Programming, ACM Computing Surveys, vol. 28, no. 4, Dec. 1996
- [7] He, H., 2003. What is Service-Oriented Architecture? <http://webservices.xml.com/pub/a/ws/2003/09/30/soa.html>
- [8] Ireland, R., 2002. What is the difference between object-oriented and service-oriented programming? [http://expertanswercenter.techtarget.com/eac/knowledgebaseAnswer/0,295199,sid63\\_gci984127,00.html](http://expertanswercenter.techtarget.com/eac/knowledgebaseAnswer/0,295199,sid63_gci984127,00.html)
- [9] Kay, A., 1993. The early history of Smalltalk, The second ACM SIGPLAN conference on History of programming languages, Cambridge, Massachusetts, United States, 1993, pp. 69 – 95
- [10] Kolonay, R.M. and Sobolewski, M., 2004. Grid interactive service-oriented programming environment, Proceedings of the 11th ISPE International Conference on Concurrent Engineering (ISPE/CE2004), Beijing, China, July 26-30, 2004, pp. 97-102
- [11] Li, B., 2004. Ontology & Service Oriented Programming, <http://asusrl.eas.asu.edu/oso/>
- [12] Momentum Software Ltd., 2004. SODA - Service Oriented Development of Applications, <http://www.serviceoriented.org/>
- [13] Potok, T.E., Vouk, M. A. and Rindos, A., 1999. Productivity Analysis of Object-Oriented Software Development in a Commercial Environment, Software – Practice and Experience, vol. 29, no. 10, 1999, pp. 833-847
- [14] Sillitti, A., Vernazza, T. and Succì, G., 2002. Service Oriented Programming and its Application in GIS Integration: a New Paradigm of Software Reuse, Proceedings of the 7th International Conference on Software Reuse, Austin, TX, 2002, pp. 269-280
- [15] Sobolewski, M., Soorianarayanan, S., Malladi-Venkata, R. K. 2003. Service-Oriented File Sharing, Proceedings of the 2nd IASTED International Conference, Communications, Internet, and Information Technology, Scottsdale, AZ, USA, November 17-19, 2003
- [16] Stevens, M., 2002. Service-Oriented Architecture Introduction, Part 1, [http://www.developer.com/net/article.php/10916\\_1010451\\_1](http://www.developer.com/net/article.php/10916_1010451_1), Oct., 2002
- [17] Stevens, M., 2002. Service-Oriented Architecture Introduction, Part 2, <http://www.developer.com/net/article.php/1014371>, Oct., 2002
- [18] The Openwings Tutorial, 2002. Service Oriented Programming Principles, [http://www.openwings.org/openwings-1.1/tutorial/Trail\\_DesignTopics/03\\_SOPPrinciples.html](http://www.openwings.org/openwings-1.1/tutorial/Trail_DesignTopics/03_SOPPrinciples.html)
- [19] Zhu, H., 2004a. Some Important Factors in Teaching Software Engineering Courses, Proceedings of the 2004 Canadian Conference on Computer and Software Engineering Education (C3SEE'04), University of Calgary, Calgary, Alberta, Canada, March 29-30, 2004, pp. 43-52
- [20] Zhu, H., 2004b. The Role Mechanism in Collaborative Systems, Proceedings of the 11th ISPE International Conference on Concurrent Engineering (ISPE/CE2004), Beijing, China, July 26-30, 2004, pp. 455-461
- [21] Zhu, H. and Chen, H., 1990. Composing Software Components Based on the Mechanisms of Smalltalk, Computer Science, vol. 8 (3), 1990 (in Chinese)
- [22] Zhu, H., Yang, G., and Liu, Z., 1998. Object-Oriented Principles and Applications, Publishing House of Changsha Institute of Technology, Sept. 1998 (in Chinese)
- [23] Zhu, H. and Zhou, M. C., 2003. Methodology First and Language Second: A Way to Teach Object-Oriented Programming”, Proceedings of the 2003 Educator’s Symposium on Object-Oriented Programming, Systems, Languages and Applications (OOPSLA’03), California, USA, Oct., 2003, pp. 140-147
- [24] Zhu, H., and Zhou, M.C., 2005. Object-Oriented Programming with C++: A Project-Based Approach, Tsinghua University Press (to be published in 2005)