Advanced Separation of Concerns for Dynamic, Lightweight Languages

Douglas R. Dechow

Oregon State University
Corvallis, OR 97330
dechow@cs.orst.edu

Abstract. The purpose of this dissertation is to investigate the use of advanced separation of concerns techniques—specifically, aspect-oriented programming—in the context of the problems that call for the use of dynamic, lightweight languages. Advanced separation of concerns techniques are feasible and useful for dynamic, lightweight languages. The AOPy project is an aspect-oriented system implemented in the scripting language Python. A common scripting language task, the development of web application services, has been identified as a representative problem domain. A technique from Discount Usability Engineering, known as a Thinking-Aloud study, will be used to evaluate aspect-oriented programming in the problem domain.

1 Rationale

Aspects and aspect-orientation [1] offer a controllable, modular mechanism for describing the separation of concerns [2] that are orthogonal to the object model that is the primary developmental focus of a wide range of software applications. Much of the research that has been done in the area of aspect-oriented programming has been focused on systems programming languages, in particular Java [3]. Other investigations in the area of advanced separation of concerns techniques have also focused on systems languages such as C++ and Java [4][5][6][7]. It is worth noting that much of the earliest work in this area was done using dynamic languages [1][8].

More colloquially known as scripting languages, the use of dynamic, lightweight languages has seen significant growth with the rise of web development [9]. The flexibility and rapid-development cycle of these languages has proven to be very attractive in this arena [10][11].

Industry-related software development is not the only area that has seen an increase in the interest of scripting languages. Academic research involving dynamic, lightweight languages continues to grow. The past two years have seen two lightweight language workshops sponsored by MIT [12][13].

Finally, an anecdote will serve to substantiate that the combination of aspect-orientation with scripting languages is a subject worthy of examination. At the 2002 OOPSLA workshop "Runtime Kernel Support for Dynamic Languages and Component Based Architectures," [14] Yukihiro Matsumoto, the creator of the Ruby scripting language, pointed out that on a recent visit to Oracle Corporation, he had

learned that an internal project made use of a Perl script (dare we say, a program?) that contained 3M lines of code.

Clearly, if advanced separation of concerns techniques offer the promise of untangling code and improving modularity, the time has come to evaluate their use in scripting languages.

2 Goals

The AOPy project is a vehicle to verify that those problems that scripting languages are used to solve would benefit from the application of advanced separation of concerns technologies. Additionally, the project will validate a specific implementation of an aspect-oriented tool—AOPy. In order to satisfy the above goals, the AOPy project will follow three specific lines of inquiry.

First, in the chosen problem domain—web application services—what are the appropriate concerns/aspects that are scattered and tangled throughout the system?

Second, while it is clear that scripting language programmers are already quite successful in this domain, does the addition of a second dimension for decomposing concerns (aspects) to the decomposition tools that are already present (function/objects) allow scripting language programmers to perform their tasks more effectively? In other words, in a comparison between objects+aspects vs. objects, does the pairing of aspects with objects suggest an improvement over plain-old objects?

Finally, can scripting language programmers effectively identify new concerns in the problem domain? Having done so, can they use AOPy to create new aspects that modularize the identified concerns?

Assessing the second line of inquiry listed above—evaluating the effectiveness of objects+aspects vs. objects—is the primary goal of this research project.

3 Methodology

The AOPy project consists of artifacts that can be divided into three groups: an aspect-oriented infrastructure, a problem domain, and an experiment design.

3.1 Aspect-Oriented Infrastructure

The software portion of the AOPy project consists of two pieces: an aspect weaver and a set of aspects. The design of the weaver is similar to the general-purpose, runtime aspect weavers described in [8][15]. The system is similar in that it is inheritance based and non-invasive. The two primary separation of concerns techniques used in AOPy's weaver are method-interception via a dynamic proxy-like system and advice based on mix-ins.

As is described in [9], the core functionality of an aspect-oriented system or language rests upon three criteria: "a join point model, a means of identifying join points, and a means of affecting implementation at join points."

The join point model used in AOPy is based on method-calls. In [16], it is argued that method-call join points are sufficient for an aspect-oriented system. Like AspectS [17], AOPy does not introduce any new language constructs to support aspect-orientation. The join point language of AOPy is Python itself. At runtime, method-interception is used to enable the insertion of advice at join points. Advice—a means for affecting implementation—is provided in the form of mix-ins.

Additionally, as mentioned above, it is necessary to develop a set of aspects that facilitate experimentation in the context of the problem domain. Some possible aspect candidates are as follows: web page user/password verification, tracing, application server logging, and persistence. The aspects will be implemented using the same language—Python—and tools as the aspect weaver.

At this time, AOPy is undergoing prototyping and active development. It is expected that the core functionality system will be demonstrated by September 2003.

Although it is too early in the system's development to make any predictions about performance, it is widely recognized that the use of reflective techniques can have adverse performance effects. Great care will be taken to minimize these effects.

3.2 Problem Domain

Initially, the area of web application services was chosen as the problem domain because of its close association with the use of scripting languages. However, support for the experiment design was another consideration in choosing an appropriate problem domain. The domain must be amenable to the examination of cross-cutting concerns. In other words, there must be a reasonable number of potential aspects for the system. Also, the domain must be pedagogically reasonable and engaging. This means that the system must be involved enough to approximate real-world situations, but not so large and complex as to be burdensome in an experimental situation.

The web application services domain satisfies these criteria. As is demonstrated by the list of possible aspect candidates in section 3.1—user/password verification, tracing, application server logging, and persistence—the domain can support a reasonable number of potential cross-cutting concerns.

The Snakelets Python web application server [18] was chosen as the software system for the problem domain because of its reasonable size and its open source license.

3.3 Experiment Design

The final artifact to be developed for this dissertation is an experiment design. Based upon advice and comments that I received from a dissertation committee member and another researcher, my claims will be validated by designing an experiment based on the Thinking-Aloud method [19]. This is an extension of the 'Simplified Thinking-Aloud' method that is a component of Discount Usability Engineering [20].

Thinking-Aloud experiments are based on three components: 1) an evaluation; 2) user participation; and 3) a diagnostic and summary [21]. Thinking-Aloud studies result in qualitative data obtained via observing the subjects participation in a problem-solving session. It is important to note that Thinking-Aloud is not a statistical methodology. A slight modification to the Thinking-Aloud methodology involves using paired subjects in the diagnostic. This method is known as a Constructive Interaction session [22]. This modification could be very beneficial when dealing with both students and professionals. It allows for testing larger numbers of students in an allotted period of time. It will also reduce the amount of time necessary to test an allotted number of professionals. Minimizing the time imposition on professionals is of paramount importance given the current industry demands.

For the specific incarnation of this experiment, two groups of developers will each attempt to design, implement, and debug a solution in the context of the problem domain. One group of developers will be working strictly within the object-oriented paradigm. The other group will be working with object-orientation augmented by aspect-orientation. As currently envisioned, the experiment will take place at the end of a course that introduces the subjects to Python, the problem domain, and advanced separation of concerns techniques (dynamic proxies, mix-ins, and aspect-orientation). In preparation for the experiments, the subjects will be exposed to object-oriented and aspect-oriented solutions for a specific set of problems. During the survey, the subjects will be given a similar set of problems with planned modifications. The subjects will then proceed to solve the problem using techniques appropriate to their group—either object orientation only or object orientation plus aspect orientation.

As an example, a specific task could be based on providing password authentication for web pages. During the course, the students would be shown how to do this using both methodologies for a single page. During the experiment, the students would then have to demonstrate their understanding of their test group's methodology by providing authentication for a larger set of web pages.

This experiment design is tightly integrated with the material presented in the course that precedes the experiment. The introductory section on the Python programming language can be modified according to the experiences of the test subjects, either students or professionals. However, the materials concerning the problem domain and advanced separation of concerns techniques will be identical for all subjects.

In keeping with the prerequisites of the Thinking-Aloud and Constructive-Interaction methods, the activities of the subjects during the user-participation section will be monitored and recorded.

At this time, it unknown exactly what will be measured. Several researchers, including the reviewers of this paper, have suggested the first group of subjects to participate in this experiment should be evaluated in an open-ended fashion. The outcomes of this first group could then be used to shape the experiment diagnostic for the next group of test subjects. This seems to be a reasonable and prudent course of action.

An initial group of test subjects (an undergraduate programming language class) has been identified. It is hoped that the experiment can be refined and repeated with graduate students and industry scripting-language programmers. A wide cross-section of experiences and skills would be beneficial for validating the project's goals.

References

1. Kiczales, G. et al.: Aspect-Oriented Programming in Proceedings of ECOOP '97. Lecture Notes in Computer Science, Vol. 1241. Springer-Verlag, Berlin Heidelberg New York (1997)
2. Hursch, W., Lopes, C.: Separation of Concerns in Technical Report NU-CCS-95-03. Northeastern University, Boston (February 1995)
3. Kiczales, G., Hilsdale, E., Hugunin, J., Kersten, M., Palm, J., Griswold, W.G.: Getting Started with AspectJ in Communications of the ACM, Vol. 44, Iss. 10 (October 2001) 59-65
4. Czarnecki, K., Eisenecker, U.W.: Generative Programming: Mathods, Tools, and Applications. Addison-Wesley, Upper Saddle, NY (2000)
5. Lieberherr, K.: Adaptive Object-Oriented Software: The Demeter Method with Propagation Pattern. PWS, Boston (1996)
6. Ossher, H., Tarr, P.: Multidimensional Separation of Concerns in Hyperspace in IBM Research Report, RC 21452(96717) (16 April 1999)
7. Tarr, P. Ossher, H., Harrison, W., Sutton, S.M.: N Degrees of Separation: Multi-Dimensional Separation of Concerns in Proceedings of International Conference on Software Engineering. ACM (1999) 107-119
8. Bollert, K.: Aspect-Oriented Programming, Case Study: System Management Application. Graduation thesis, Fachhochschule, Flensburg (1998)
9. McMillan M.: Scripting Languages Go Primetime in Software Development Magazine. (April 1998)
10. Ousterhout, J.: Scripting Higher-Level Programming for the 21st Century in IEEE Computer, Vol. 31, Iss. 3 (March 1998) 23-30
11. Martin, R.: Are scripting languages the wave of the future? http://www.itworld.com/AppDev/1262/itw-0314-rcmappdevint/
12. Lightweight Languages Workshop. http://ll1.ai.mit.edu
13. Lightweight Languages Workshop. http://ll2.ai.mit.edu
14. First International Workshop on Workshop Runtime Kernel Support for Dynamic Languages and Component Based Architectures. http://hosting.msugs.ch/auroraborealis/Workshops/LAR02/Lar02Home.htm
15. Bollert. K.: Implementing an Aspect Weaver in Smalltalk. Invited Lecture at Smalltalk and Java in Industry and Education. (1997)
16. Ossher, H., Tarr, P.: Operation-level Composition: A Case in (Join) Point in ECOOP '98 Workshop Reader. Lecture Notes in Computer Science, Vol. 1543. Springer-Verlag, Brussels (1998)
17. Hirschfeld, R.: AspectS—AOP with Squeak in OOPSLA Workshop on Advanced Separation of Concerns in Object-Oriented Systems. http://www.prankinf.tu-ilmenau.de/~hirsch/Projects/Squeak/AspectS/
18. The Snakelets Web Application Server. http://snakelets.sourceforge.net
19. Jorgensen, A.: Using the thinking-aloud method in system development. In: Salvendy, G., Smith, M.: Deisgning and using human-computer interfaces and knowledge based systems. Elsevier Science, Amsterdam (1989)
20. Nielsen, J.: Usability engineering at a discount. In: Salvendy, G., Smith, M.: Deisgning and using human-computer interfaces and knowledge based systems. Elsevier Science, Amsterdam (1989)
21. Thinking Aloud. http://sunrize.nada.kth.se/usor/jml.cgi/Methods/thinking.jml?graphics=true
22. O'Malley, C.E., Draper, S.W., Riley, M.S.: Constructive Interaction: A method for studying human-computer-human interaction in Proceedings of the International Federation for Information Processing. North Holland, Amsterdam (1984) 269-274