Extensions of Design Patterns to Control Autonomous Objects

Daisuke SUZUKI

School of Information Science,
Japan Advanced Institute of Science and Technology

February 13, 1998

Keywords: Pattern Oriented, Design Patterns, Distributed, Cooperative, Software Development, Autonomous Objects.

In this paper, we propose a software development style using domain specific *Design Patterns* extended to construct a distributed software development environment using *autonomous objects*. We also describe usability of that extended Patterns.

We generally take the following way to develop a software using frameworks and kits. (W.M.Tepfenhart described in "A Unified Object Topology")

1. Analyze a domain to make the domain model.
2. Decide and Design an architectural style to realize the model.
3. Design an architecture based on the architectural style and implement the framework.
4. Build kits to fill the hot spots of the framework.
5. Construct an application with the framework and kits.

There are, however, large gaps between 1 and 2 and between 3 and 4 which may cause difficulty to map each other. This large gap should be filled by architectures. Recently, we take a development style using *Design Patterns* (E. Gamma) to make a framework or kits flexible and reusable, which is called "Pattern Oriented Programming".

Design Patterns have a number of good features like reusable designs or easiness to understand. But Design Patterns are also have some problems. Sometimes it is called as *Micro Architecture*, and it is very small piece of design. This property causes serious problems to design a whole system architecture. Because it does not show the way how to apply a number of Design Patterns to the system architecture, and how to select suitable Design Patterns. In addition to these, Design Patterns have some problems that it cannot apply to implementation easily and it doesn't consider a domain and implementation language. Design Patterns are not contain a domain specific or language specific knowledges.

In this paper, we propose a software development style using extended or modified *Design Patterns* to solve the already mentioned problems and to make the best use of Design Patterns. We analyze one of target domains to find some issues of Design Patterns and extension or modification policies to adjust to the domain. For this purpose, we choose for the target domain to "Distributed and Cooperative Software Development Environment on Computer Network (aka JIZAI)" that is currently developed in our OCHIMIZU-laboratory at JAIST. "JIZAI" is a model that defines an information repository that keeps the state of artifacts (products) and decisions, reflecting the state of progress precisely. And it is expressed as "A number of autonomous objects, and it's execution environment". We design and implement some small-scale programs that precisely express the functions that is needed for "supporting environments for Autonomous objects". Applying the selected Design Patterns to that small-scale programs, we find that how to extend, how to modify the Design Patterns, and what patterns are needed to create.

Now we define the *"Autonomous Objects"*. A general definition of an autonomy is defined by M. Jackson. He classifies into five categories for the definition as following;

- *Active* : This can do something without any impulse from outer world. Active can be divided in three sub category; *Autonomous* : This cannot be controllable from outer world. So this cannot be a computer software. *Biddable* : This can control from outer world. (This can do something actively only if it was ruled.) *Programmable* : This can work something only by programmed rules.
- *Reactive* : This can only react upon impulses from outer world.
- *Inert* : This cannot do anything automatically.

Our "Autonomous Object" is categorized as *Programmable* on his categories. But it may do something autonomously in cooperative work. Because if someone works with my object, it may change its own states, and this change cannot control by me. So, we call it "Autonomous Objects".

Then, we design five small-scale programs that examines the following functions:

1. Observable objects which is observed by another objects.
2. Objects can pull out another object references which is created or destroyed randomly.
3. Object can composite to another objects dynamically. This function may be designed as specification of Java language.

4. Object can have a *State Transition Diagram*, and it should be encapsulized. This function may be used with first function. This program should build as extension of first program.

5. Referring to distributed objects can be transparently.

The five small-scale programs are designed as simpler enough to construct a program.

Now we sort out an issue of Design Patterns, and define the extension policies as below by analyzing above software designing and programming.

- Encapsulate an object referring process to hide dispersion of objects.
- Consider synchronous message communications or object referring.
- Consider repercussions among with objects to give activeness for a system.
- Consider a method of pattern combination to design entire architecture easily.
- Consider an autonomy of objects.
- Accept a Java specific design method or implement method aggressively to provide an implementation of patterns.

On the design policies as stated above, we define some extended patterns that are needed for “JIZAI” domain with designs and implementations of above small-scale prototype programs. And we also propose some newly patterns to make up for the deficiency of Design Patterns. Using this extended patterns as a whole system architecture that is considered as implementation or domain model, we can develop a software (application, framework, kits) more easily. Since we specify a target domain (as distributed software development environment) and implementation language (as Java), we can offer a some template code as implementation. And we also examine an availability of the software development style with extending or modifying Design Patterns. Then we apply the extended patterns to “JIZAI” system to design a whole system architecture. We finally show an availability of the extended patterns.

Now we are challenging to develop a prototype system architecture of “JIZAI” using our extended patterns to investigate the usability of the patterns. And also we are examining some issues of extended patterns caused by our modifications, and also examining an applicability of the extended patterns to another domains or another frameworks. And we need to examine the extended patterns to refine. After I design a prototype system architecture, I will feed back experiences to my extended patterns to refine them.