Using Osprey for

Understanding Object-Oriented Software

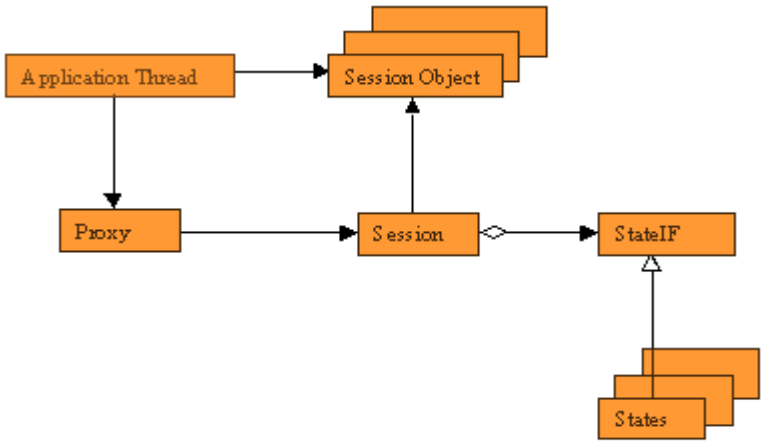
MITRE-Sponsored Research

Project Number: 51MSR102-A3

Principal Investigator: Penny Chase

Task Lead: David Harris

Project Staff: Angel Asencio, Sam Cardman, Ellen Laderman, Suzi Lubar, Scott Mardis



Introduction

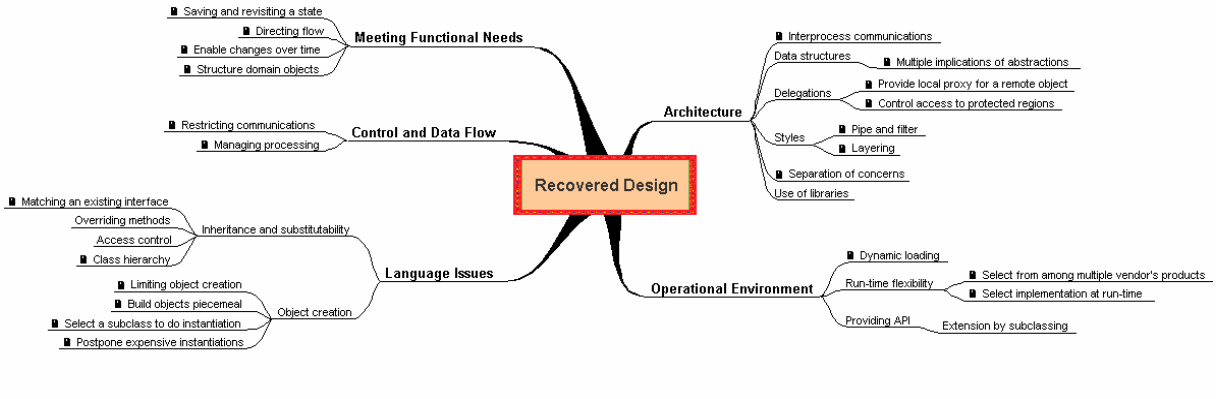
In the spring of 2000, we were asked to help with an assessment of object-oriented software. A MITRE sponsor needed to assess the quality of a C++ and Java software system. Specifically, the sponsor wanted to know if the software could be reused. This need was typical of the many needs that motivate analyses of software. Examples

include maintenance questions, reuse questions, code upgrades, component integrations, and analysis of hidden and explicit requirements.

We knew that commercial and open source resources for analyzing object-oriented software were limited. There were few tools available that fully parsed C++ or Java and these tended to be very expensive. Additionally, due to indirection caused by inheritance and polymorphism, it is difficult for these tools to correctly compute data flow and control flow.

Although the use of object-oriented languages is becoming pervasive, the indirection mentioned above and the lack of suitable tools make analysis an imprecise, error-prone art form. Thus, we recognized that the development of tools to understand object-oriented software would be an important area for research and development.

The obvious candidates for understanding object-oriented software structures are design patterns. Members of a design pattern community have developed best-practice descriptions for creating object-oriented software. These descriptions are agreed upon knowledge that bridges between software constructs, their rationale, and their consequences for code quality. If we know that developers used a particular pattern (intentionally or implicitly), we have a good idea of what forces (see those in the figure as examples) influenced the artifact's designers and how they chose to harness those forces in the design and implementation.



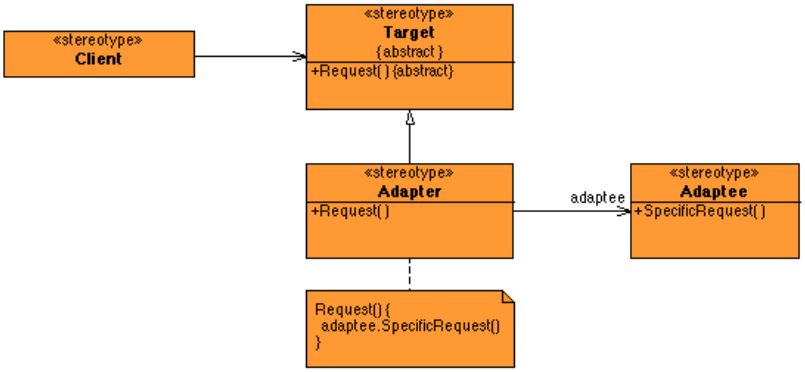
Forces influencing design

We developed a prototype software system named **Osprey** [1]**.** To identify occurrences of design patterns in the software, we developed 104 recognizers – declarative descriptions of the constraints that must be met if one is to conclude that a pattern is present in the code. Then, given a program to analyze, Osprey matches the relationships among that program's structures and the pre-built design pattern templates.

For example, the *Adapter* design pattern is used to adapt a new service to meet the interface of a pre-existing set of services. One might have a set of services that works with x-y coordinates, but the program needs to take advantage of a service that produces results in distance-and-bearing coordinates. The way to achieve this is to use an *adapter*. A key component of the *Adapter* pattern is that there must be some mechanism for the

[1] Osprey stands for Object-oriented Software Pattern REcoverY

adapter to know what is being adapted. Hence, Osprey identifies potential adapters that know about other structures and change the results that are generated by these other structures – in our example, changing distance-and-bearing results to x-y coordinates.



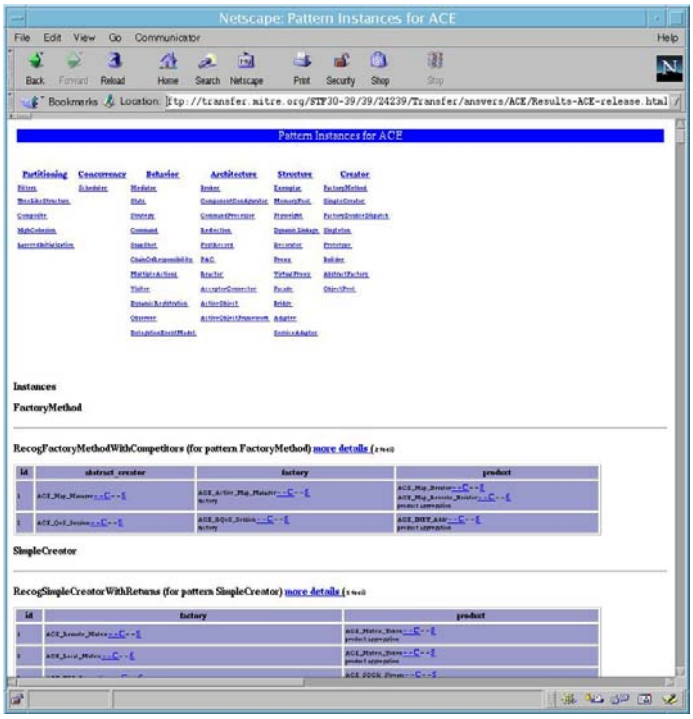
Adapter Pattern Template

With Osprey, analysts can look at programs in a more informed way. They are able to make statements such as "The purpose of class A is to adapt the results produced by class B in order to match the interface defined by class C." They can also make statements about code quality. We identified source code qualities (e.g. reliability, evolvability,and performance) effected by the use of design patterns and added this information to Osprey. For example, the presence of the pattern *AbstractFactory* supports maintainability.

Osprey generates HTML documentation based on all of the analysis it does on the source code. The generated documentation consists of:

- code-level reports (classes, methods),
- recovered design patterns,
- software quality and design rationale inferred from the use of patterns, and
- mappings from high-level expectations down to the source code that meets those expectations.

Osprey provides a way for analysts to access data through design concerns and design patterns. One page enumerates all the pattern participants for the pattern instances in the code (see the figure below for a partial view of the design pattern recognition results).



Design Pattern Web-page

Using Osprey Output

Analysts can start with program structure, with design pattern instances, with quality issues, with vulnerability concerns, or with design concepts. From any of these starting points, analysts can navigate to design patterns and to color-coded source code files. The table below lists the reports Osprey is able to generate, their category, a brief description of their contents, and the source language(s) for which they are generated.

Analysts who want an in-depth understanding of how the program is structured can start with design patterns. Analysts who wish to assess the software quality for source selection, reusability, or interoperability can start with quality reports or perhaps reports that show use of operating system services. Analysts who need to assess the potential vulnerabilities of the code can start with any of a collection of vulnerability reports.

Reports on the Source Code

Report Name	Category	Description	Source Language
Executive Summary	Summary	Summary of source code statistics and major observations	All
Class Details	Source Code Information *and* Recovered Design Patterns	Class information including: methods, inheritance, and pattern participation	C++, Java
Class Hierarchy	Source Code Information	Class inheritance hierarchy	C++, Java
Class Instantiation	Source Code Information	Who creates a class and classes it creates	C++, Java
Methods	Source Code Information	Properties of the program's methods	C++, Java
Struct Details	Source Code Information	Lists of structs defined in the program	C
Procedures	Source Code Information	Properties of the program's procedures	C
Operating System Calls	Source Code Information	List of operating system calls	C, C++
Pattern-based Qualities	System Qualities	Links to program quality information based on pattern usage	C++, Java
Pattern Alternatives	System Qualities	Pattern impacts on program qualities and alternative pattern suggestions	C++, Java
Inheritance Faults	System Qualities	Potential variable initiation flaws due to polymorphism	C++, Java
Design Goals	Design Analysis	List of potential design goals and supporting pattern usage	C++, Java
Design Facts and Evidence	Design Analysis	Facts about the source code design	C++, Java
Design Index	Design Analysis	Index into pattern descriptions	C++, Java
Potential Vulnerabilities	Security Concerns	Programming constructs that could lead to security violations	C++, C
Vulnerabilities by File	Security Concerns	File security evaluation	C++, C
Port Access	Security Concerns	List of ort numbers referenced in the source code	C++, C
Vulnerable OS Calls	Security Concerns	List of vulnerable OS calls used in program being analyzed	C++, C
Pattern Deviations	Security Concerns	Deviant implementations of design patterns	C++, Java
Pattern Instances	Recovered Design Patterns	List of pattern instances	C++, Java
Patterns within File Structure	Recovered Design Patterns	Pattern participation organized by file	C++, Java
Pattern Diagrams	Home	Basic pattern descriptions	C++, Java

Conclusions

Our research program makes progress in two areas of assisting analyst/users in software understanding tasks. We developed recognizers that identify instances of

software design pattern use in source code and we applied inference mechanisms to recovered information in order to provide insight into the design process itself. We have demonstrated the utility of design pattern recovery and the feasibility for automatic recovery of design rationale and software qualities of object-oriented software.

Additional Information

Osprey is available for use in analysis of software systems – either as a service we provide or as a tool that can be used at your site.

Readers who would like to know more about Osprey technology may wish to read *"Relating Expectations to Automatically Recovered Design Patterns"* by A. Asencio, S. Cardman, D. Harris, and E. Laderman, published in the Proceeding of the Working Conference on Reverse Engineering, Richmond, Virginia, 2002 and available from the authors.

We can be reached by email at osprey@mitre.org.