Facilitating Pattern Repository Access with the Implicit Culture Framework

Aliaksandr Birukou, Enrico Blanzieri, and Paolo Giorgini
University of Trento, Italy
{aliaksandr.birukou,enrico.blanzieri,paolo.giorgini}@dit.unitn.it

Michael Weiss
Carleton University, Canada
weiss@sce.carleton.ca

Abstract

Software patterns enable an efficient transfer of design experience by documenting common solutions to recurring design problems. However, given the steadily growing number of patterns in the literature and online repositories, it can be hard for non-experts to select patterns appropriate to their needs, or even to be aware of the existing patterns. We describe an implicit culture approach for supporting developers in choosing patterns suitable for a given problem. It consists in providing developers with recommendations based on the history of decisions made by other developers regarding which patterns to use in related problems. The proposed architecture comprises observers that collect information about developer actions, an inductive module to develop a theory about actions performed in different situations, and a composer that suggests actions based on the theory.

1 Introduction

More than a decade ago, the authors of the book *Design Patterns* [1], the first major publication on software patterns, stated the problem of selecting patterns:

> With more than 20 [*indeed, a very small number from today's perspective*] design patterns in the catalog to choose from, it might be hard to find the one that addresses a particular design problem, especially if the catalog is new and unfamiliar to you.

Since then patterns have become a staple of current software development approaches. However, the problem of selecting patterns still exists. Moreover, it has become more severe, as the number of documented patterns is continuously increasing: for instance, the *Pattern Almanac* [2] lists more than 1200 patterns. And in the seven years since its publication, many new patterns and books on patterns have been published. The problem of choosing the appropriate pattern is particularly hard to solve for inexperienced programmers [3]. Hence, tool support for assisting in the pattern selecting process becomes of utmost importance.

Figure 1: The pattern extraction process.

In this paper, we address the problem of selecting patterns from a *social point of view*. To help a developer make a decision about which patterns to use, getting suggestions from her group of peers is important. We present an approach that helps developers choose the patterns suitable for a given problem. Basically, the approach consists in providing developers with recommendations which are created from a history of previous user interactions with the repository. The approach enables the distribution of knowledge about the use of patterns within a community of developers *without* their direct involvement. We have prototyped the proposed approach, and preliminary results are available elsewhere [4]. For the tests we used a Lucene-based implementation of the repository that contains a set of security patterns published on patternshare.org [5], one of several popular online repositories for patterns.

2 Background

Recently, there have been several efforts in making patterns available in pattern repositories, where they can be browsed and searched by various criteria. An early example was the *Pattern Almanac* [2], which is available in electronic form (`www.smallmemory.com/almanac`). A more recent example is the PatternShare site hosted by Microsoft (`patternshare.org`). (As of the date of this writing, the PatternShare site is no longer operational.) In order to store patterns in a repository, a structured pattern representation must be adopted. There have been several proposals, most notably the Pattern Language Markup Language (PLML) [6].

In this work we have adopted a format specific to a set of security patterns published on patternshare.org [5]. We have defined an XML representation for these patterns and extracted the content of the subset of this repository from the website. The extraction process is illustrated in Figure 1. Our current representation contains the following elements: Pattern.Context, Pattern.Problem, Pattern.Solution, Pattern.KnownUses, and Pattern.RelatedPatterns, as well as elements specific to the patternshare.org site, but not required for our purposes. However, our approach does not depend on a specific pattern representation.

We are also not concerned, at this stage of development, with how easy it is to deploy our approach; however, in the future; we plan to converge towards a standard like PLML.

3 Implicit Culture

This section presents an overview of the general idea of the implicit culture framework and Systems for Implicit Culture Support (SICS) that constitute the core of our approach. A more thorough description of the IC-Service, which is a general-purpose, domain-independent rec-

ommendation service implementing the implicit culture approach, can be found in [7], and a detailed description of the implicit culture theory and the SICS architecture in [8].

Our motivation for adopting an implicit culture approach stems from the difficulty less experienced developers face in using patterns. Developers who wish to apply patterns from a domain that is not their main area of expertise encounter similar difficulties. A good example is the security domain. For any but trivial applications, security is a key concern. However, making the application secure is not the main concern of the application developer. Security patterns [9] provide guidance to non-experts in security for designing secure application.

However, a significant challenge remains: how do developers decide which patterns they should use? According to [3], only experienced software engineers with a "deep knowledge" of patterns can use them effectively. This is so, because they can recognize the generic situations where a pattern can be applied in their design work. However, inexperienced programmers will find it difficult to decide whether to reuse a pattern or develop a custom solution.

The difference between these two types of developers is that an experienced developer uses implicit knowledge (in particular, her own experience) about the problem (see [10] for a more general discussion on this point). When we look at the community of a developer's peers, knowledge is considered *implicit* when it is embodied in the capabilities and abilities of the community members (developers). It is *explicit* when it is possible to describe and share it through documents or knowledge bases. To select appropriate patterns inexperienced developers should acquire the implicit knowledge that more experienced developers have.

We argue that it is possible to change the pattern selection behavior exhibited by inexperienced developers to converge with the behavior of more experienced developers by suggesting patterns suitable for their current design task. In order to determine which patterns are suitable, we use the history of the developers' previous interactions with the system, i.e. which patterns other developers have chosen in a similar situation as faced by the developer. We refer to the pattern selection behavior of experienced developers as the *culture* of this developer community. When inexperienced developers start behaving in agreement with the culture of the community, a knowledge transfer from experienced to inexperienced developers has occured. The relation characterized by this knowledge transfer is called an *implicit culture*.

Consider a developer who needs to improve access control in a system that offers multiple services. Suppose that for an experienced developer it is apparent to use the Single Access Point pattern. If the system is able to use previous history to suggest that the novice should use Single Access Point, and she follows that recommendation, then we say that she behaves in accordance with the community culture and the implicit culture relation is established.

4 Architecture

The general architecture of SICS consists of the following three components:

- an *observer*, which stores information about the actions performed by users (members of the community) in a database of observations;
- an *inductive module*, which analyzes the stored observations and uses data mining to develop a theory about actions performed in different situations;

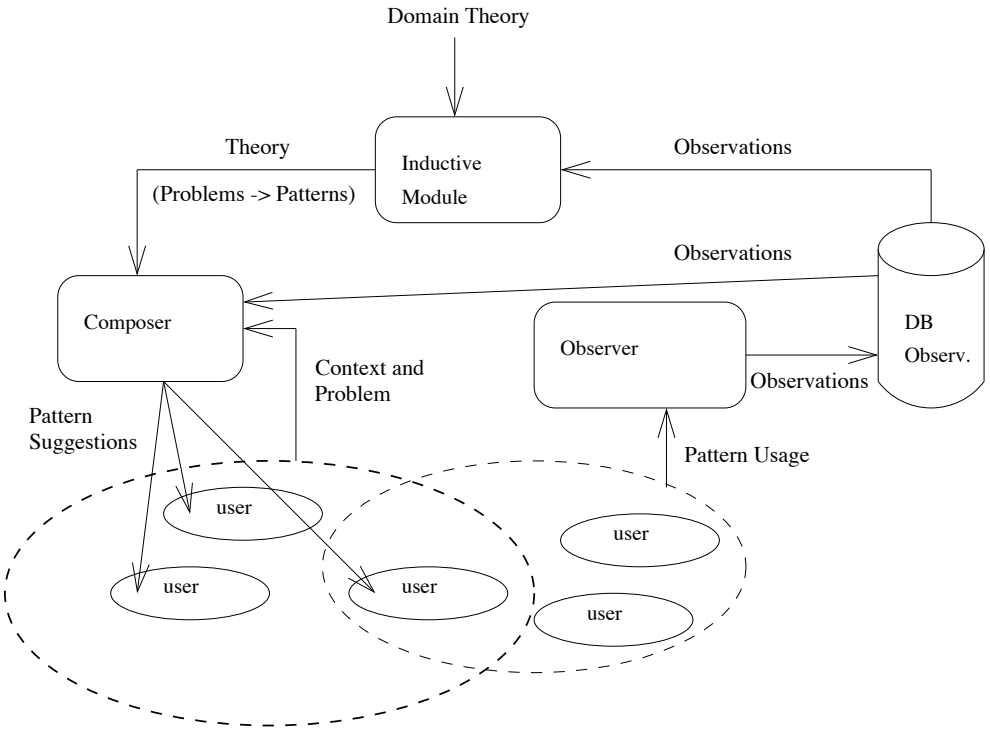

Figure 2: Architecture of SICS as applied to the patterns domain.

- a *composer*, which exploits the information collected by the observer and derived by the inductive module to suggest actions in a given situation.

In terms of our problem domain, the observer saves information about the problem and characteristics of the project, in whose context the problem occurred; which patterns have been proposed as a solution; and which pattern has been chosen in return. The inductive module discovers problem-pattern pairs (which patterns are selected for what kind of problems) by analyzing the history of the interaction of users with the system. A set of problem-pattern pairs form a *theory*. The goal of the composer is twofold. Firstly, it compares the description of a problem faced by a developer with the problem part of the theory mined by the inductive module in order to suggest the corresponding pattern. Secondly, the composer tries to match the problem with the pattern by analyzing the history of observations and calculating the similarity between the problem description given by the user and the problem descriptions which users provided for patterns previously selected for similar problems. Figure 2 summarizes the components of the proposed architecture and their interactions.

This is the basic idea behind using an implicit culture approach for selecting an individual pattern. However, it is clear that patterns are rarely used in isolation, but rather they are selected in the context of other patterns that have already been applied (see [11] for an overview of this general procedure). This means that we can also exploit links between patterns (the information in the Pattern.RelatedPatterns section of our pattern representation) to recommend patterns. In this paper, however, we have focused on the selection of individual patterns in the understanding that we will be able to use this as a step of a larger, iterative process.

5 Acknowledgements

This work is partly funded by research projects EU SERENITY "System Engineering for Security and Dependability", MEnSA "Methodologies for the Engineering of complex software Systems: Agent-based approach" and by Fondo Progetti PAT, MOSTRO "Modeling Security and Trust Relationships within Organizations" and QUIEW "Quality-based indexing of the Web", art. 9, Legge Provinciale 3/2000, DGP n. 1587 dd. 09/07/04.

References

[1] Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design patterns: elements of reusable object-oriented software. Addison-Wesley Longman Publishing Co., Inc. (1995)

[2] Rising, L.: The Pattern Almanac. Addison-Wesley Longman Publishing Co., Inc. (2000)

[3] Sommerville, I.: Software engineering (7th ed.). Addison-Wesley, Boston, MA, USA (2004)

[4] Birukou, A., Blanzieri, E., Giorgini, P., Weiss, M.: A multi-agent system for choosing software patterns. Technical report, University of Trento (2006)

[5] Hafiz, M., Johnson, R.E.: Security patterns and their classification schemes. Technical report (2006)

[6] PLML: http://www.hcipatterns.org/plml+1.0.html (2003)

[7] Birukou, A., Blanzieri, E., D'Andrea, V., Giorgini, P., Kokash, N., Modena, A.: IC-Service: A service-oriented approach to the development of recommendation systems. In: Proceedings of ACM Symposium on Applied Computing. Special Track on Web Technologies. (2007)

[8] Blanzieri, E., Giorgini, P., Massa, P., Recla, S.: Implicit culture for multi-agent interaction support. In: Proceedings of the 9th International Conference on Cooperative Information Systems. (2001) 27–39

[9] Schumacher, M.: Security Engineering with Patterns Origins, Theoretical Model, and New Applications. Number 2754 in LNCS. Springer (2003)

[10] Dreyfus, H.L., Dreyfus, S.E.: Mind over machine: the power of human intuition and expertise in the era of the computer. The Free Press (2000)

[11] Alexander, C., Ishikawa, S., Silverstein, M., Jacobson, M., Fiksdahl-King, I., Angel, S.: A pattern language. Oxford University Press (1977)