CHOOSING THE RIGHT DESIGN PATTERN: THE IMPLICIT CULTURE APPROACH

Aliaksandr Birukou, Enrico Blanzieri, Paolo Giorgini
Department of Information and Communication Technology
University of Trento, via Sommarive 14, 38050 Povo (Trento), Italy
e-mail: {aliaksandr.birukou, enrico.blanzieri, paolo.giorgini}@dit.unitn.it

ABSTRACT

Design patterns represent reusable solutions to problems that have been proved to be useful in different contexts. An experienced programmer can choose a suitable pattern for a given problem effectively. However, for an inexperienced programmer this is a very hard task. We propose a multi-agent system that supports programmers in choosing the design pattern suitable for a given problem. Personal agents in our system produce knowledge transfer among users, allowing for the reuse of experience in choosing design patterns.

INTRODUCTION

"With more than 20 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." (Gamma et al., 1995). This shows that selecting a design pattern was a problem even ten years ago. The continuously growing number of patterns (e.g, W. F. Tichy's catalog at `http://wwwipd.ira.uka.de/tichy/patterns/overview.html` contains over 100 patterns) complicates the problem even more. For an inexperienced programmer it is very hard to choose the right design pattern and tools assisting in this process become of the utmost importance.

Unfortunately, there is still a lack of systems that guide a programmer in the selection of design patterns (see (Kung et al., 2003; Gomes et al., 2003) for examples of such systems). Most of the current approaches dealing with patterns suppose that it is the programmer who makes the choice of the pattern (Ó 'Cinnéide and Nixon, 2001).

This paper addresses the problem of design pattern selection and proposes an approach that considers the problem from a social point of view. We propose the use of a multi-agent system based on the Implicit Culture framework to help programmers in selecting patterns by getting suggestions from the group. In the system, the problem faced by the programmer is compared with those faced previously by colleagues and suggestions about the most suitable design patterns are provided.

IMPLICIT CULTURE FOR DESIGN PATTERN SELECTION

This section presents an overview of Implicit Culture (IC) and Systems for Implicit Culture Support (SICS) (see (Blanzieri et al., 2001) for more details). It also contains the formalization of the problem of design pattern selection.

"Only experienced software engineers who have a deep knowledge of patterns can use them effectively. These developers can recognize generic situations where a pattern can be applied. Inexperienced programmers, even if they have read the pattern books, will always find it hard to decide whether they can reuse pattern or need to develop a special-purpose solution." (Sommerville, 2004). The difference between two programmers is that the experienced programmer uses implicit knowledge (in particular, his experience) about the problem. Knowledge is called *implicit* when it is embodied in the capabilities and the abilities of the community members. It is *explicit* when it is possible to describe and share it through documents and/or information bases. To select suitable design patterns the inexperienced programmer should acquire the implicit knowledge of more experienced programmers.

We argue that it is possible to shift the behavior of inexperienced programmers in design pattern selection towards the behavior of experienced programmers by means of suggesting patterns which are more suitable for the current design task. To determine appropriate patterns we use the history of previous interactions with the system, i.e. what patterns have been chosen for similar design tasks. We call the behavior of experienced programmers related to the pattern selection a *community culture*. When inexperienced programmers start behaving similarly to the community culture the knowledge transfer (performed by the SICS) occurs. The relation characterized by this knowledge transfer is called *"Implicit Culture"* (Blanzieri et al., 2001).

For example, let us consider a programmer that needs to define an interface for creating an object, but let subclasses decide which class to instantiate. Let us suppose that for an experienced programmer it is obvious to use Factory Method pattern here. If the system is able to use previous history to suggest that the novice uses Fac-

tory Method pattern and he actually uses it, then we say that he behaves in accordance with the community culture and the IC relation is established.

The general architecture of a SICS consists of the following three components: the *observer*, which uses a database of observations to store information about actions performed by the user; the *inductive module*, which analyzes the stored observations and applies learning techniques (namely, Data Mining or Machine Learning) to develop a theory about actions performed in different situations; the *composer*, which exploits the information from the observer and the inductive module to suggest actions in a given situation.

In terms of our problem domain, the observer saves information about the problem, which patterns have been proposed as a solution and which pattern has been finally chosen. The inductive module discovers problem-solution pairs by analyzing the history of users' interaction with the system. A set of problem-solution pairs form a *theory*. The goal of the composer is to compare a problem faced by the programmer with the problem part of the theory and to suggest the corresponding solution part. If this step fails, the composer tries to match the problem with the solution by calculating the similarity between the descriptions of the problem and the patterns.

To formalize the problem of pattern selection for a community of programmers we need to decide how to describe patterns and design problems and how to match a given design problem with an appropriate pattern.

According to (Gamma et al., 1995) a design pattern has the following four parts: the *name*, the *problem*, the *solution*, and the *consequences*. The most interesting part for us is the *problem*, since it contains the description of the task solved by the pattern. For instance, in the catalogue of GoF patterns, the information about the problem is contained in the section "Intent". The system should match this description with a description of the design problem. We assume there exists a (semi)formal or textual description of the problem faced by the programmer and of the problem solved by the pattern. In IC terms these problems constitute *situations*. Each situation is associated with *actions* performed by *agents* (programmers) on *objects*. The actions can also have *attributes*, which are features that can be useful for the analysis. In our application, the SICS analyzes the actions presented in Table 1 (see (Birukou et al., 2006a) for more details). Every action is recorded as being performed in a context of a certain project, characterized by an optional attribute *project_name*. We introduce this attribute, because a project that has some specific requirements can influence the choice of patterns. The goal of the SICS is to offer the programmer actions which have been performed in similar situations, i.e. to suggest the use of patterns that have been chosen for solving similar problems.

Table 1: Actions that can be Observed by the System

action	objects	attributes
request	problem_description	project_name
apply	pattern, problem_description	project_name
reject	pattern, problem_description	project_name

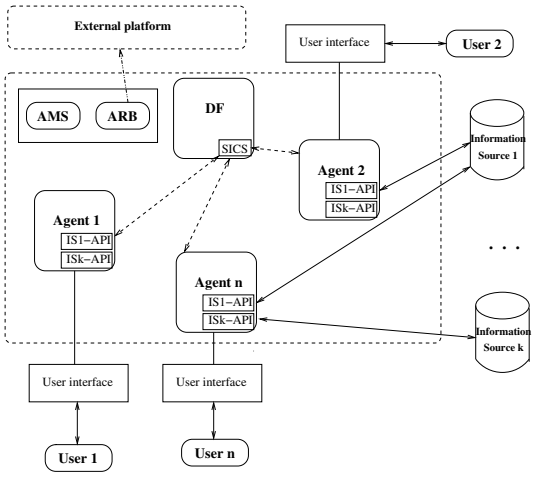

Figure 1: Architecture of IC Multi-Agent Platform.

USING IMPLICIT CULTURE MULTI-AGENT PLATFORM FOR PATTERN SELECTION

In this section we briefly describe the architecture of the multi-agent system for design pattern selection.

We start from the description of a general IC Multi-Agent platform which was previously applied to the problem of web link recommendation for a community of users (Birukov et al., 2005) and to scientific publications search (Birukou et al., 2006b). The platform is implemented using the Java Agent DEvelopment framework (JADE. `http://jade.tilab.com/`). The general architecture is depicted in Figure 1 and it consists of the following components (see the JADE documentation or (Birukou et al., 2006a) for more details). A *user* submits requests to the system using the interface on the client side. A request contains the description of the design problem. A *Personal Agent* (PA) is a software agent which assists the user in choosing a suitable pattern. It uses dedicated API to query several *information sources*, which contain information about design patterns. The *Directory Facilitator* (DF) provides agents with the IDs of other PAs and uses the SICS module to produce recommendations about patterns that have been selected for similar problems. The recommendations are based on previous experience of the users.

A typical usage scenario is as follows: a user submits a request, expressing a design need. The personal agent contacts the DF. The DF stores the request action in the database of observations and uses the SICS module to find patterns chosen for similar problems previously. The suggestions are sent to the personal agents and are shown to the user. Moreover, it is also possible to show

descriptions of similar problems discovered, because this can help the user to take a decision. Finally, being able to see a list of similar problems faced by his colleagues, the programmer could have a short talk with them to share their experience.

EVALUATION AND DISCUSSION

We are currently working on the implementation of information sources. We use Apache Lucene (`http://lucene.apache.org/`), an open source search engine library, to create an index of the pattern repository and to search for pattern descriptions using this index. So far we have built a repository of 23 design patterns from (Gamma et al., 1995). Personal agents use API to access Lucene searching capabilities. We have not performed the system evaluation with real users yet. However, in (Birukov et al., 2005) we presented numerical results obtained using the simulator developed for the application of the IC Multi-Agent Platform to web search. The results have shown that an increase in the number of users causes an increase in the recall of the suggestions produced by the system. We think that the problem of design pattern selection is much related to the problem of selecting web links relevant to keywords and that our approach will be also useful here.

The Expert System for Suggesting Design Patterns (ESSDP) (Kung et al., 2003) and REBUILDER (Gomes et al., 2003), similarly to our system, suggest design patterns to solve problems faced by designers. There are several differences between our approach and ESSDP. Firstly, ESSDP assumes knowledge acquisition (human experts must fill in the knowledge base) as the primary step. Differently, in our system the SICS learns from the interactions with users (both expert and novices), without any initial knowledge base, allowing for continuous improvement of suggestions. Secondly, our architecture is not restricted to the use of a rule-based knowledge base assuming that different learning techniques can be adopted. REBUILDER uses Case-Based Reasoning (CBR) techniques for the pattern selection. The main differences between IC and CBR are the following: while CBR solves a new problem by remembering a previous similar situation and by reusing information and knowledge of that situation, IC helps to solve a problem but does not produce the solution directly; in CBR, the problem is represented explicitly (the case), while IC does not deal with an explicit representation of the problem, only with the implicit information about it.

The system can be extended in a number of ways. For instance, it might also be applied to the selection of the most appropriate version of the pattern, when several versions are available. A semiformal representation of the design problem (e.g., class or activity) or a formal framework, such as the non-functional requirements (NFR) framework (Gross and Yu, 2001) can be adopted for the description of design problems/design patterns.

We also think that the problem we address is related to the problem of web service composition, where to solve a sub-problem it is necessary to find a web service which suits some design need (Lazovik et al., 2006). Our approach can be also used in tools for refactoring the old code using patterns, e.g., (Ó 'Cinnéide and Nixon, 2001), at the stage of selecting the pattern: a programmer can be provided with suggestions about patterns used in similar situations previously.

CONCLUSION

A system that helps programmers in choosing design pattern suitable for a given task has been described. The system takes into account the social part of the problem, providing users with suggestions from other community members. As future work we would like to conduct several experiments with real users.

ACKNOWLEDGEMENTS

This work is funded by EU SERENITY, COFIN, MOSTRO and QUIEW. We would like to thank the anonymous reviewers for their helpful comments.

REFERENCES

Birukou, A., E. Blanzieri, and P. Giorgini. 2006a. Choosing the right desing pattern: an implicit culture approach. `http://eprints.biblio.unitn.it/archive/00000960/`. Technical report.

Birukou, A., E. Blanzieri, and P. Giorgini. 2006b. A multi-agent system that facilitates scientific publications search. In *AAMAS '06*, 265–272: ACM.

Birukov, A., E. Blanzieri, and P. Giorgini. 2005. Implicit: An agent-based recommendation system for web search. In *AAMAS '05*, 618–624: ACM.

Blanzieri, E., P. Giorgini, P. Massa, and S. Recla. 2001. Implicit culture for multi-agent interaction support. Volume 2172 of *LNCS*, 27–39: Springer.

Gamma, E., R. Helm, R. Johnson, and J. Vlissides. 1995. *Design patterns*. Addison-Wesley.

Gomes, P., F. C. Pereira, P. Paiva, N. Seco, P. Carreiro, J. L. Ferreira, and C. Bento. 2003. Selection and reuse of software design patterns using cbr and wordnet. In *SEKE'03*.

Gross, D., and E. S. K. Yu. 2001. From non-functional requirements to design through patterns. *Req. Eng.* 6 (1): 18–36.

Kung, D. C., H. Bhambhani, R. Shah, and G. Pancholi. 2003. An expert system for suggesting design patterns: a methodology and a prototype. In *Software Engineering With Computational Intelligence*, ed. T. M. Khoshgoftaar. Kluwer Int.

Lazovik, A., M. Aiello, and M. Papazoglou. 2006. Planning and monitoring the execution of web service requests. *Journal on Digital Libraries*. To appear.

Ó 'Cinnéide, M., and P. Nixon. 2001. Automated software evolution towards design patterns. In *IWPSE '01: Proc. of the 4th Int. Workshop on Principles of Software Evolution*: ACM.

Sommerville, I. 2004. *Software engineering*. Addison-Wesley.