An Intelligent Tutor for Teaching Software Design Patterns

LUIS BERDUN,[1] ANALIA AMANDI,[1,2] MARCELO CAMPO[1,2]

[1]ISISTAN *Research Institute, Faculty of Sciences,* UNICEN *University, Campus Universitario,* B7001BBO *Tandil, Buenos Aires, Argentina*

[2]CONICET, *Argentina*

Received 15 April 2011; accepted 6 October 2011

ABSTRACT: How to teach students to design in the classroom? When is experience crucial to do design? In particular, how to teach design patterns to students who are beginning to know the importance of a good design? Experience is essential to understand and apply patterns in an effective way. Generally, novice users are not good at working in real experiences while they are good at learning new techniques and methods for designing. In this work, we show the results of teaching patterns using an artificial intelligent assistant that helps novice developers during the design process. Our assistant is an interface agent that observes novice users working, and when it detects that a design pattern can be applied, it makes a suggestion justifying its opinion. Thus, students understand when and where a pattern could be applied. © 2011 Wiley Periodicals, Inc. Comput Appl Eng Educ 22:583–592, 2014; View this article online at wileyonlinelibrary.com/journal/cae; DOI 10.1002/cae.20582

Keywords: intelligent agents; teaching design patterns; intelligent tutor

INTRODUCTION

Teaching design patterns is a challenging task. Not only it is necessary to communicate the solution structure offered by a design pattern, it is also essential students understand when and where a pattern could be applied [1].

In the last few years, pattern catalogues have become a commonplace in the development and design of applications and frameworks. These catalogues have shown to be very useful to capture flexible and reusable object-oriented structures, promoting also a shared vocabulary for developers to talk about object-oriented design issues. Undoubtedly, design patterns represent one of the most important contributions to design practice, but paradoxically, to understand and, what is yet more important, apply them in a productive way require much design experience. In other words, inexperienced designers tend to overuse or misuse knowledge of patterns producing poor quality designs.

Experience at teaching design patterns has revealed that simply picking out a particular pattern from 'the gang of four book' [2], or similar text, and presenting it in the well-known format (intent, motivation, applicability, structure,..., known uses) is ineffective to teach people who lack design experience and typically cannot see the real value of a pattern. Often the adequacy of a given pattern to solve a problem in the best way is more a matter of common sense than knowledge of patterns themselves. Novice designers, who are exposed to patterns for the first time, should engage in real problems where patterns can help in order to understand that a given solution can exist and be applied [3]. In general, the students are often unable to comprehend patterns from isolated examples [4].

Biggs remarks that activities that have a significant and positive impact on learning include talking things through with others, using and doing things in real life, and teaching others [5]. It is well known that reading does not stimulate high-order thinking since students remain as passive recipients of information, then teaching patterns only by the exclusive use of traditional reading of patterns catalogues could be a problem.

Apart from learning individual patterns and the principle behind them, students should learn how to understand and apply patterns they have not seen before, how to integrate different patterns and how to use this knowledge in real-life situations [6].

According to this, Goldfedder and Rising [7] highlight a remark taken from a Personal Communication with Ralph Johnson that they proved to be right after their experience with a beta course: '... people cannot learn patterns without trying them out. Also, people need to find them in their own problem domain'.

An example of this, but applied for teaching drawing-based conceptual design skills, is the work presented in Ref. 8

Correspondence to L. Berdun (luis.berdun@isistan.unicen.edu.ar).

where preliminary tests show how using an education-oriented computer application improves the capacity for spatial vision.

An approach satisfying these requirements would involve a student solving a real problem (in a domain that he/she knows) and a tutor making suggestions about the application of a pattern (if any). The application of this approach, however, could be very hard because it requires a high number of tutors and real problems tend to be too large and complex to be implemented within the time available in an undergraduate course. Therefore, automated assistance complementing a CASE[1] tool to help the novice developer to understand, learn and apply design patterns becomes a valuable help in the teaching process.

In particular, we argue that an interesting approach to approximate the previous goal is the use of *interface agents* [9]. We base this argument on the fact that these agents have been designed to assist human users. Basically, the main idea is to have a specialised entity monitoring the user's activities, detecting opportunities to help his/her work, and then, offering guidance on how specific tasks should be carried out. Interface agents have been used to assist students in other subjects such as personalised assistance to e-learning [10], also as intelligent tutors at different domains SQL-Tutor [11]; German Tutor [12]; a prelab tutoring system [13] to helps students to get acquainted with laboratory instruments as well as experimental procedures for the Strength of Materials experiment; ITSPOKE [14] in the domain of physics problems; CIRCSIM-Tutor [15], who helps students solve a class of problems in cardiovascular physiology dealing with the regulation of blood pressure; and KERMIT [16], who teaches conceptual database design using the Entity-Relationship data model.

In the teaching of design patterns these agents are able to help students during their design process like a personal tutor. We envision the following: (i) a student modelling some design problems through a CASE environment; (ii) the specification of diagrams within the CASE; (iii) our agent, designed to recognise situations where patterns can be applicable, can suggest the pattern solution (if any) that best suits the current design context. To enact with the above scenario, we have improved a CASE tool with an intelligent agent called *PatternAdvisor*. This agent observes the work that is being developed by a student and suggests, according to this, which patterns could be useful to apply. The suggestion is accompanied with information about the pattern, their benefits and problems, and how to modify the student's design to apply the pattern. The suggestion always tries to improve the user solution using a design pattern; in the proposal the user classes to be modified are included as well as how to modify them. In general, there may be several suggestions for the same user design; this could be the same pattern several times, but with different ways to be applied in the user design, or different design patterns. Because all suggestions are linked with the pattern explanation, the user may select the best way of modifying his/her design. The suggestions include a full explanation of the design pattern, including benefits and problems of its application.

With this assistance, the students learn how patterns can be applied to their work. They experiment the whole process of a pattern, from definition to instantiation, learning how the pattern can help them to improve their design.

[1]Computer-aided software engineering, software used for the automated development of systems software.

The rest of the article is organised as follows. Section 2 describes the main characteristics of our *PatternAdvisor* agent. Section 3 introduces how *PatternAdvisor* can assist students with respect to design patterns. Section 4 discusses some experiences and experiments carried out to test the tool with students. Finally, section 5 outlines the conclusions of the article as well as some lines about future work.

AN OVERVIEW OF THE *PatternAdvisor* AGENT

The main topic of this article is the use of an intelligent tutor for teaching patterns. To understand this we need to know how the *PatternAdvisor* agent works. First we explain the *PatternAdvisor* behaviour and its general operation, after that we will focus the explanation on the relation between the agent and novice users.

The *PatternAdvisor* is an interface agent that collaborates with a user using a CASE tool, through a visual interface that allows the user to interact with the agent when it has some suggestion to make. As mentioned, we have two different types of users, expert developers and novice developers (our students). *PatternAdvisor* assists students in relation to the design pattern application using the knowledge obtained by the observation of expert designers and the pattern catalogue.

PatternAdvisor is provided with knowledge of pattern catalogues represented through a Bayesian Network (BN). A Bayesian Network is a compact, expressive representation of uncertain relationships among variables of interest in a domain. A BN is a directed acyclic graph that represents a probability distribution, where nodes represent variables and arcs represent probabilistic correlation or dependency between variables [17]. The strengths of the dependencies are given by probability values. For each node, a probability table specifies the probability of each possible state of the node given each possible combination of state of its parents. These tables are known as conditional probability tables (CPT). Tables for root nodes (or independent nodes) just contain unconditional probabilities. This network is appropriate to represent causal relationships among design patterns and design tasks. These relationships have been defined from the individual analysis of each design pattern and an analysis of the relations among patterns in the pattern catalogue. Also, the agent's knowledge base can be enhanced through the observation of the design actions that an expert takes to incorporate design patterns into a design specification. In this way, *PatternAdvisor* follows the steps involved in the correct application of design patterns. This information is used to update the probabilities of the theoretical BN allowing the representation of heuristic knowledge of the application of design patterns. In such a way that our agent follows the steps involved in a *right* application of design patterns, although the developer is not aware of this situation.

At some point in time, *PatternAdvisor* will have accumulated enough evidence, and it is determined that some pattern(s) may be of utility to restructure the classes in the diagram. When doing so, the agent observes a novice developer when he/she performs tasks such as: creation of classes and sequence diagrams, addition/removal of classes and objects, drawing of class relationships and message flows, edition of names, attributes or operations, among others. All these actions help to determine a potential application context for pattern solutions basing the decision on the Bayesian Network.

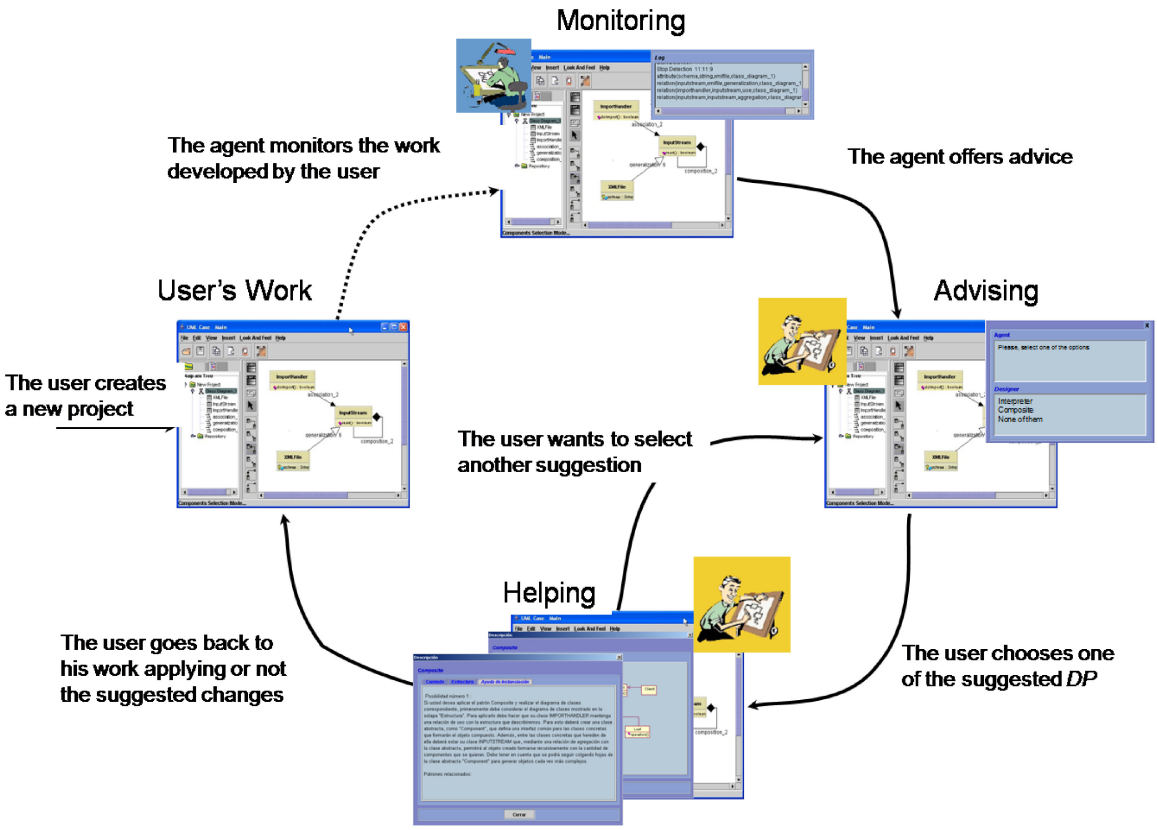


Figure 1 The work phases of *PatternAdvisor* agent. [Color figure can be seen in the online version of this article, available at http://wileyonlinelibrary.com/journal/cae]

Figure 1 shows the lifecycle of the *PatternAdvisor* agent when it is working with a novice user. Here, a developer interacts with a traditional CASE application, editing different UML[2] diagrams of his/her project. Also, this figure shows the different interfaces and dialogs displayed by the agent; the agent colour is blue when it is monitoring the user actions and is yellow when it has some advice to give. It is important to point out that the agent usually does its work in an autonomous way, although the user can still require the agent's guidance explicitly or ignore the agent.

The lifecycle starts when the novice user creates a new project. The agent observes him/her while he/she is performing tasks such as: creation of classes and sequence diagrams, addition/removal of classes and objects, drawing of class relationships and message flows, edition of names, attributes or operations, among others. These observations are used and gathered as evidence. When the *PatternAdvisor* will have collected enough evidence, and so it will be able to decide that some pattern(s) may be of high utility to (re)structure the classes in the diagram.

At this point, the agent starts a conversation with the user in order to offer advice (Advising phase in Fig. 1). As an outcome, the user will receive a list of recommended patterns. He/she may consult one of the patterns, prefer to get an alternative pattern or simply turn down the suggestions and proceed with the design on his/hers own.

If the user selects a design pattern from the list of suggested patterns (Helping phase in Fig. 1), the agent will provide an explanation for the pattern, including class and sequence diagrams, and the needed actions to transform the current user's design to include the suggested pattern. Also, it is feasible for the agent to propose more than one alternative to apply the suggested design pattern; this is because sometimes there are different ways to apply a given pattern. It is always the user who decides to apply or not the changes suggested by the agent.

Figure 2 shows the conversation between the agent and the student. In this case the conversation has been initiated by the agent. The boxes contain the sentences said by the agent and the arrows are labelled with the options selected by the user. In box number 3 of Figure 2, we can appreciate how the agent suggests a pattern and the required changes to apply this pattern in the current work.

ASSISTIVE FEATURES

In terms of assisting novice users, the goal of assistant agents is the identification of which patterns could improve the student's project. Basically, the *PatternAdvisor* monitors the work developed by the user and searches for evidence that suggest that applying a pattern to improve the current design could be

[2]Unified Modelling Language is a standard notation for the modelling of real-world objects as a first step in developing an object-oriented design methodology [16].

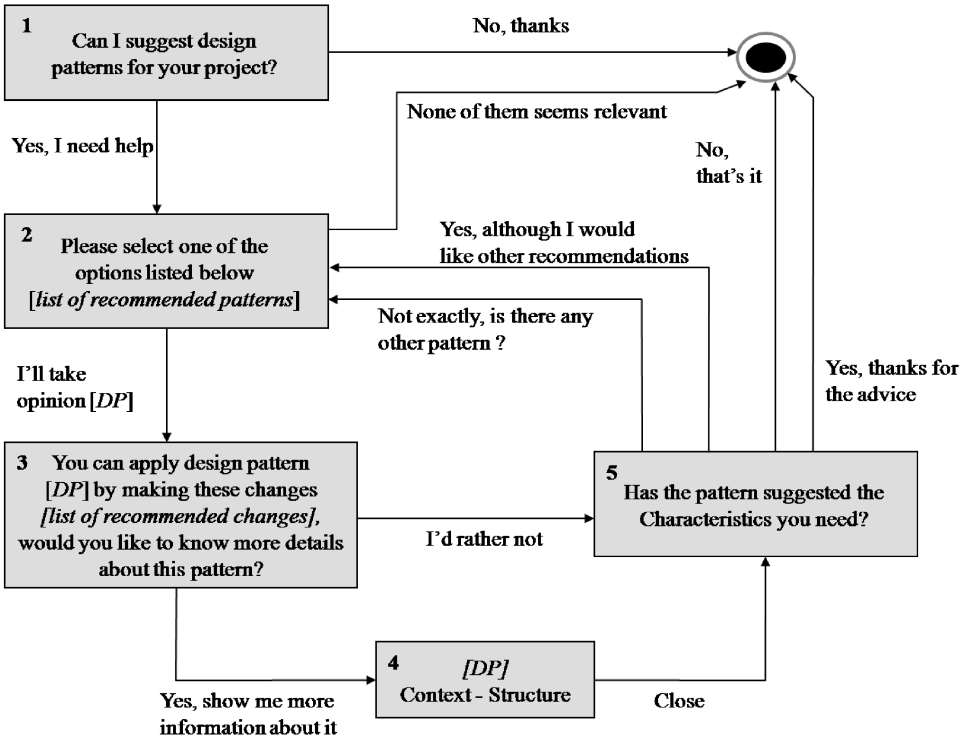


Figure 2 Conversation between the agent and the developer.

useful. The agent registers each action of the user in the case tool and transforms the actions in evidence. For example, if the user adds a new class called *aClassName*, the agent registers *class(XX)*, where XX is the variable which represents the new class. After that the user adds a relation of uses between the class *aClassName* with itself, then the agent registers *uses(XX,XX)*.

The agent uses as core of the assistance process a BN. It keeps tracks of the changes in the different available UML diagrams in the project, and transfers them to its BN representation. Afterwards, it calls a specific algorithm in order to infer a list of candidate design patterns. The patterns on this list can be actually thought of as the 'most probable' patterns for the actual design context.

Monitoring the User

In this section, a rather simple example based on the Composite pattern is presented with the goal of emphasising the mechanisms used by the agent and not the pattern complexity. As defined, the aim of the Composite pattern is to compose objects in tree structures to represent part-whole hierarchies. This pattern lets clients deal with individual objects and compositions of objects uniformly [2]. This knowledge is used to guess the user's intentions, and give him suggestions on the application of the pattern.

The scenario starts when a user begins to design his/her solution to the given problem using the CASE tool. In Figure 3a, we can appreciate the first approach developed. These classes, relations and operations are analysed by the agent in order to gather some evidence that suggests the possibility of applying a pattern. The analysis of this first approach evidences that the Composite pattern could be applied to the current design because the user shows interest in working with a composition of objects. While the user adds classes, relations or operations, the agent can find new evidence that confirms or refuses its hypothesis.

In the example, the user adds several elements originating Figure 3b. In this situation the agent maintains the first evidence as true, and adds new evidence about the application of the Composite pattern. This is because now it is easier to see that the user wants to work with a composition of objects. Also, in this case the classes Rectangle and Line represent evidence that the user desires to treat individual objects and compositions of objects uniformly (class Picture).

Moreover, the class diagrams are not the only way used by the agent to discover evidence. It uses sequence diagrams as another way to find new evidence. This kind of diagrams gives information about the dynamicity of the design developed by the user. Because of this, the agent can learn how the user intends to use the classes that are present in the class diagram.

Figure 4 shows two different sequence diagrams developed by the user. These diagrams add more evidence of the applicability of the Composite pattern. For example, Figure 4a shows an object *C* (instance of class *Client*) sending the message *aOperation()* to the object *PA* (instance of class *Picture*) and how this object delegates some part of its behaviour to the object *PB* (instance of class *Picture*) using the same method *aOperation()*. On the other hand, Figure 4b shows how the functionality of an object *Pa* is carried out among the composed functionality of several objects (simple objects or composed objects).

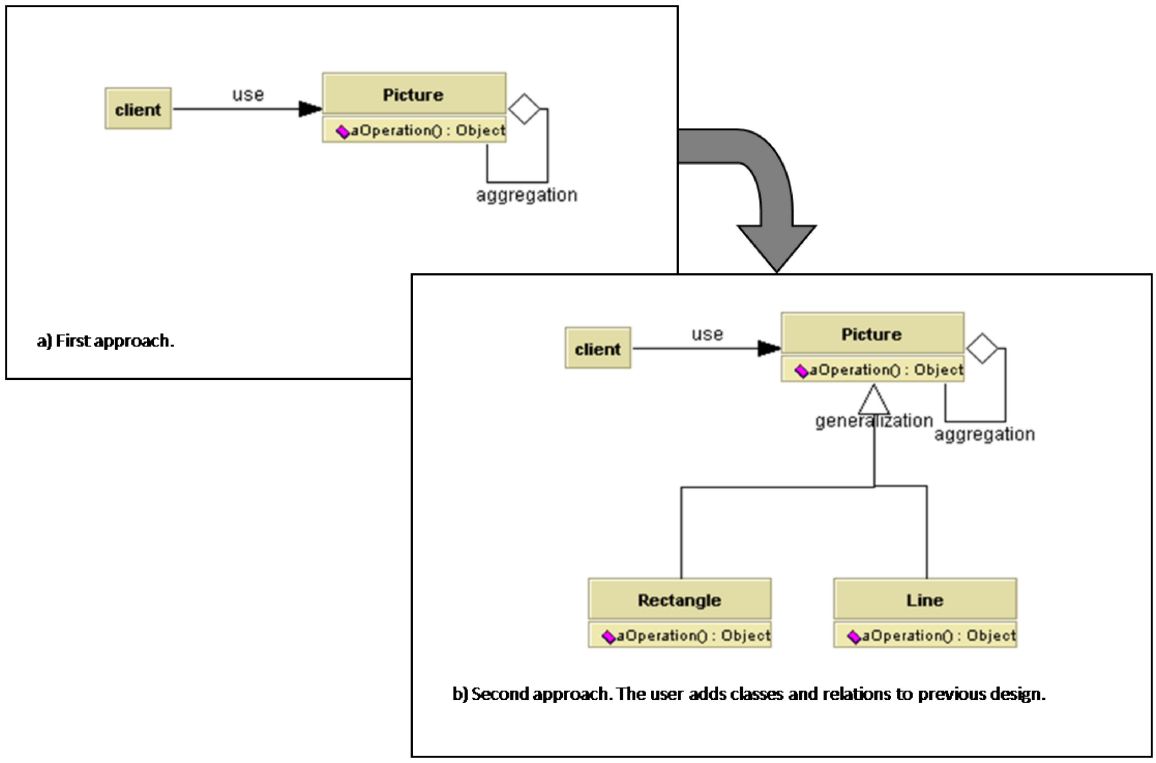


Figure 3 Class diagrams developed by the user using the CASE tool. [Color figure can be seen in the online version of this article, available at http://wileyonlinelibrary.com/journal/cae]

Advising the User

The previous example shows how, from the work done by the user, the agent searches evidence to make suggestions on the possibility of applying design patterns. Likewise, it is necessary to point out that evidence represents only subsets of the work developed by the user. In other words, to suggest a pattern, only the presence of evidence in some part of the user's work is necessary. With this mechanism the agent can suggest the application of several patterns in different places of the design. The latter allows the user to understand the individual pattern and how the patterns can be composed to do a better job.

Moreover, the names assigned by the user to the classes or methods are not used by the agent at the moment of detecting evidence (this may sound obvious, but sometimes the names of

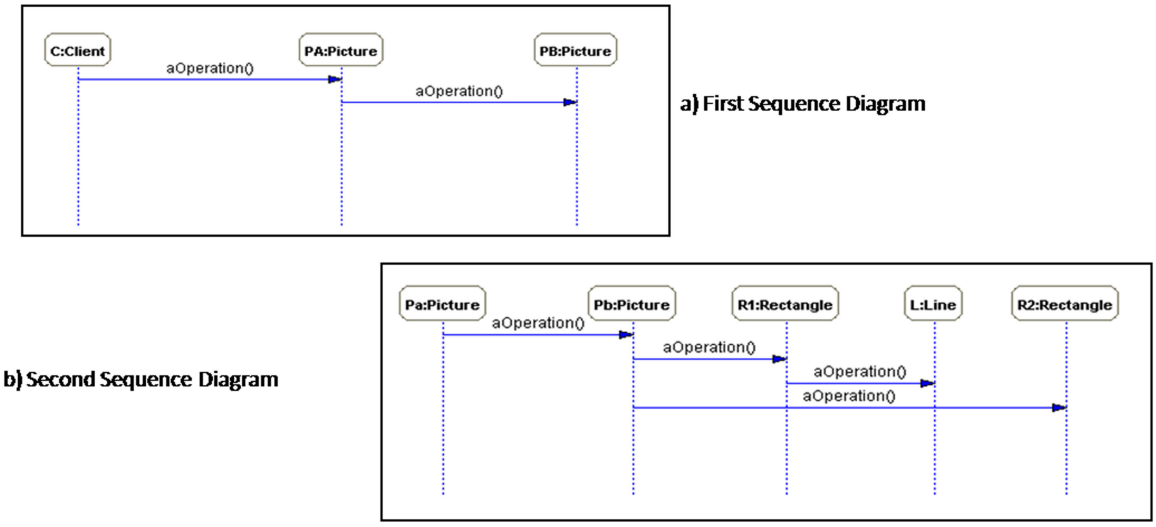


Figure 4 Sequence diagrams developed by the user in the CASE tool. [Color figure can be seen in the online version of this article, available at http://wileyonlinelibrary.com/journal/cae]

the classes can be used as evidence, for example classes named '*single*' and '*compose*' can evidence the composite pattern). Instead, the agent searches the structure of the evidence. For example, in Figure 3b it only asks for the equivalence of the methods, that is that the method present in the class *Picture* is the same as that present in the class *Rectangle* and the class *Line*.

Each piece of evidence is a variable of the BN and stands for a set of Prolog rules. For instance, a Prolog-like characterisation of an application context for this pattern would look like the one shown in Figure 5. For a better understanding of the rules that define this context, we have additionally included the visual counterparts of these rules.

Each time a predicate evidence ... (_,_) holds, the agent will have a new argument for the applicability of the associated patterns. In general, the specification of rules relies on a collection of predefined Prolog predicates, which can be checked as the UML diagrams stored in the CASE tool.

In Figure 5 we can see the specification of two common pieces of evidence to apply both, the *Composite* and *Interpreter* patterns. The first evidence in the figure is related to a sequence diagram. In this case, we check the existence of a method that invokes objects that are playing a particular role. If these restrictions are fulfilled on the sequence diagram, we have evidence about the application of the Composite pattern.

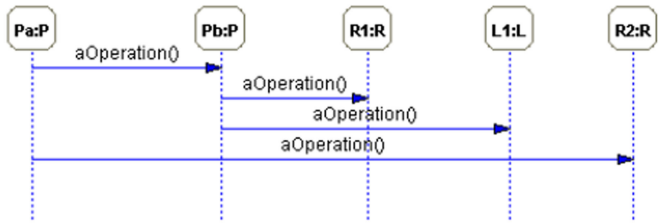
The second evidence is related to class structure on a specific diagram. In such a diagram, three classes with any name are instantiated in the variables *Client*, *Expression* and *TerminalA*. Then, we look for some relationships like a use relationship between *Client* and *Expression*, an aggregation relationship in *Expression*, and an aggregation or generalisation relationship between *Expression* and *TerminalA*. Then, we control the existence of the same operation in the class *Expression* and the class *TerminalA*. If these relationships are found, we have one piece of evidence about the possibility of applying the Composite pattern or the Interpreter pattern. In this case, these two patterns could be put forward because this class evidence is used to detect both of them.

In this example, according to the diagrams developed by the user, the *PatternAdvisor* collected enough evidence and it suggests the Composite pattern. In this case, the agent suggests only one pattern, but depending on the context it could suggest more than one.

Helping Users

Finally, the agent shows how the user could improve his/her design by applying a given pattern, in our example the Composite one. The agent shows information about the pattern (structure, intend, etc.) and the list of changes needed to apply the recommended pattern.

```
evidenceS11(P,R,L) :-
        object (Pa, P, DSequence),
        object (Pb, P, DSequence),
        not(Pa == Pb),
        object (R1, R, DSequence),
        not (P == R),
        object (R2, R, DSequence),
        not(R1 == R2),
        object (L1, L, DSequence),
        not (P == L), not (R == L),
        message (Pa, Pb, AOperation, DSequence),
        message (Pb, R1, AOperation, DSequence),
        message (Pb, L, AOperation, DSequence),
        message (Pa, R2, AOperation, DSequence).
```




```
evidenceC20(Client,Expression,TerminalA) :-
        class(Client, concrete, DClass),
        class (Expression, TypeClass, DClass),
        not (Client == Expression),
        class (TerminalA, concrete, DClass),
        not (TerminalA == Client),
        not (TerminalA == Expression),
        relation (Client, Expression, use, DClass),
        relation (Expression, TerminalA, TypeRelationAG,
        DClass),
        validateRelationAG (TypeRelationAG),
        relation (Expression, Expression, aggregation, DClass),
        operation (Analize,Type,Expression, DClass),
        operation (Analize,Type,TerminalA, DClass),
        same(Analize, Expression, Analize, TermninalA).
```

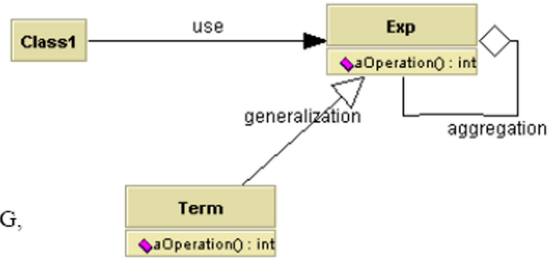


Figure 5 Evidence used by the agent to detect patterns. [Color figure can be seen in the online version of this article, available at http://wileyonlinelibrary.com/journal/cae]

The agent only suggests the list of changes; it does not propose the automatic transformation for two primary reasons: first, it is necessary for the user to understand the pattern and how it should be applied in the current design; and second, it is feasible for the agent to suggest several alternatives to apply the pattern. Moreover, the automatic transformation of the user's work does not stimulate users to learn about the suggested pattern.

Learning From the Expert User

Previously we mentioned that the notion of 'applicability' should be interpreted in terms of 'a higher probability of' rather than 'a definitive advice for' the target pattern. The question is, when should the agent make a suggestion about the applicability of a pattern? Even if the agent can detect if a piece of evidence of a pattern is true or false, the problem arises when defining the moment it gathers enough evidence to suggest the applicability of a pattern.

The use of a Bayesian Network allows us to work with uncertainties. Moreover, the agent watches and learns from the expert user with the goal of adjusting the suggestion thresholds. As this user discards or accepts suggestions about the applicability of a given pattern, it is feasible to calibrate the BN parameters. This mechanism allows adjusting the suggestions to the expert user criteria.

In this way, the expert user defines when the agent should suggest a given pattern. This training is obtained by the agent in a non-intrusive way. The agent uses the responses given by the expert, in the dialog with the agent (e.g. Fig. 2) to calibrate the BN parameters. With this training, the expert user could influence the agent suggestion, putting emphasis on the determination of patterns.

Using the tool, it is possible for a different expert to calibrate the BN parameters changing the emphasis of the patterns according to his/her belief. There is not collision between different experts because a different training re-calibrates the BN adapting the emphasis to the new knowledge.

EXPERIENCES

We have carried out some tests to prove the impact of using the *PatternAdvisor* agent in the learning of design patterns. We evaluated the behaviour of our agent with a set of 120 System Engineering students. As a requirement of the course Object Oriented Programming they had to solve one exercise individually using the tool improved with the *PatternAdvisor*. In order to vary the topics of the exercises we divided the students into four groups of 30. The evaluation was made in the Computers Labs of the Faculty.

Previous to each experiment we explained briefly Design Patterns to the students and gave to them some examples of their application. We made this in order to involve the students with background knowledge about patterns. In general, the exercise was a simple OOP problem that they could solve using previous knowledge, but the solution was a known pattern design. With this mechanism we tried to prove the utility of *PatternAdvisor* during the whole process of learning patterns.

Environment Setup

Before the experiments, we adjusted the precision of the pattern advisor. To do this we tested the assistance in different scenarios (around 15 classes, objects and relationships). After that, we evaluated the agent behaviour when it was exposed to several groups of patterns (we use scenarios with 4, 5 and 6 patterns together). This evaluation involves the possibility of feedback with respect to the usefulness of the recommendations; at this stage we use the tool in expert mode. Table 1 shows the results for one of the evaluated scenarios. Each test incorporates gradually expert developer's feedback (test1 has no feedback). The precision can be seen as the number of patterns correctly detected divided by the total number of patterns actually identified. The recall can be seen as the number of patterns correctly classified divided by the total number of patterns that should have been identified. F-Measure combines the previous indicators and it is calculated as $2^{*}\text{recall}^{*}\text{precision}$, the F-measure should tend to 1 as long as the agent becomes more proficient with its recommendations. This table shows the evolution of the precision, recall and F-measure indicators in the evaluated scenario.

The evaluated scenario was composed by 8 classes, 8 relations, 3 attributes and 11 methods.

Student Experiences

In general, the results were positive and the student did not feel invaded by the agent. On the contrary, the users explicitly asked the agent for suggestions, this was because they were expecting for a suggestion.

During the first test, the students asked the agent for a suggestion with only a few classes without relationships, waiting for the complete solution from the agent. To minimise this attitude we explained that the agents helped them with their design process and did not solve problems. After this explanation they understood that they had to solve the problem and then, during this process, they could be helped by the agent.

Figure 6 shows an example of suggestion made by the agent. The figure shows the design made by a student and how the pattern advisor suggests a modification to his/her design. The suggestion includes a list of possible patterns; for this example the patterns are *Bridge*, *Flyweight* and *Adapter*. When the student selects a pattern from the list, the agent shows the options for the selection. In the figure, the student selected the pattern *Adapter* and the Agent shows: (1) the context of the pattern, with an explanation, examples, and the benefits and problems; (2) the structure of the pattern, a class diagram and a sequence diagram explaining the pattern; (3) the way of applying the pattern in the user design. At this stage the agent presented how to adapt the user design to include the pattern, as it is highlighted in the figure remarks, the explanation is personalised to the student design.

One of the features that the students most appreciated was the possibility of knowing about the context of the pattern, and examples. Table 2 shows a summary of both positive and negative issues obtained from the experiments with the students.

Table 1 Evolution of the Measures According to Expert Developer's Feedback

	Test1	Test2	Test3
Precision	0.86	0.91	1
Recall	0.77	0.88	1
F-measure	0.74	0.92	1

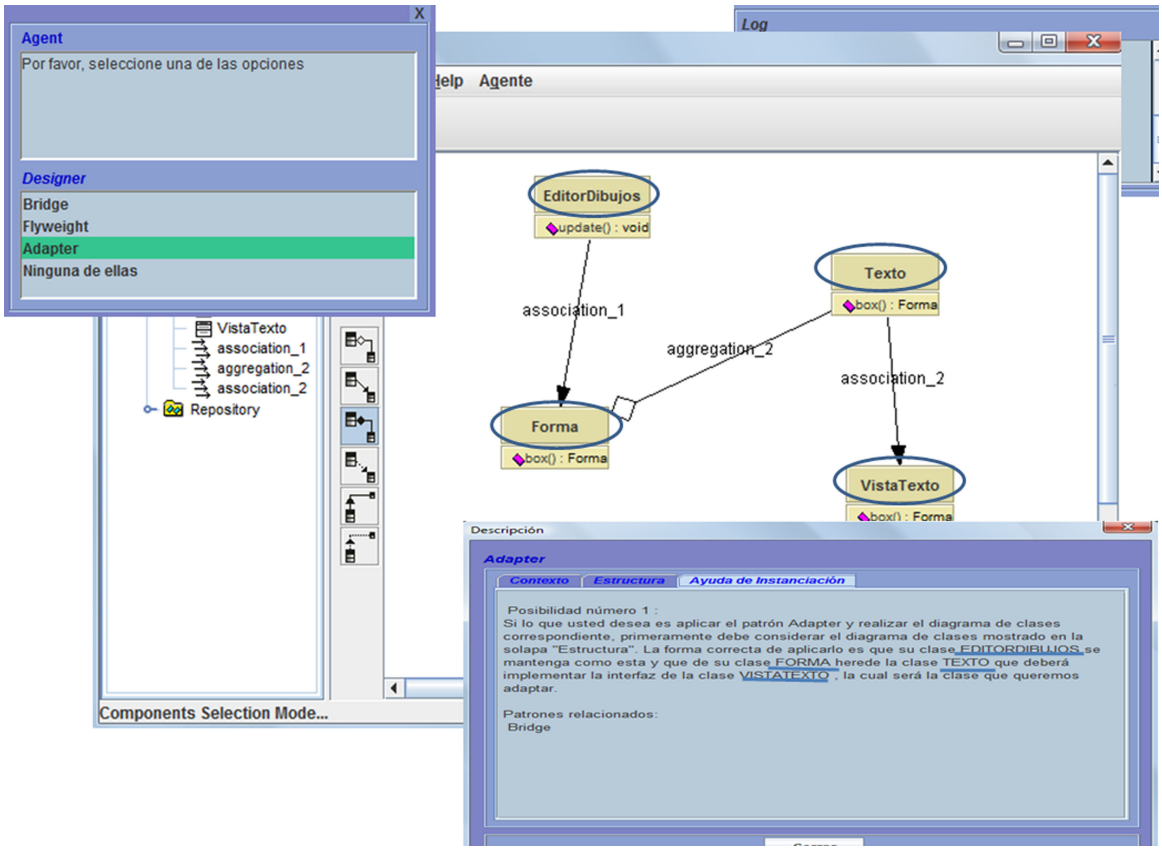


Figure 6 An example of suggestion made by the agent. [Color figure can be seen in the online version of this article, available at http://wileyonlinelibrary.com/journal/cae]

In several cases, some students applied a pattern and waited for a suggestion from the *PatternAdvisor*. This was a mistake because the agent suggests applying a pattern to improve the design and it does not inform the user that he/she has applied a pattern. In the next session we discuss these questions.

As a result from these experiments we could appreciate different positive aspects:

- Students do not feel invaded. They do not feel *PatternAdvisor* as an evaluator, and they have collaborative behaviour.
- *PatternAdvisor* as an active pattern catalogue. Students use the knowledge of *PatternAdvisor* to clear up their doubts.
- Suggestion during their task. Students see how *PatternAdvisor* suggests modifications to their designs; they do not see the agent's suggestion as an abstract one, they appreciate how their design could be improved applying a pattern design.
- Common language. After the evaluations the students show a common knowledge of patterns. This difference was notorious with other students from previous years. They understand the *context*, *benefits* and *problems* of applying a pattern.
- The pattern as a proven solution to a particular problem. This could be one of the most important achievements when using *PatternAdvisor*; they understand that the pattern is a solution to a particular problem and not to all possible problems.
- In previous years, students tried to apply patterns to all the possible scenarios and they believed that more patterns were equivalent to better designs. Now, they understand that a pattern could improve some aspects of their design, but that could worsen other aspects.

CONCLUSIONS AND FUTURE WORK

In this article, we have outlined an approach to assist the novice developer in the application and learning of design patterns. This approach is based on an interface agent called *PatternAdvisor*. It analyses class and sequence diagrams edited by the developer in a CASE tool, and gives advice on those pattern solutions that seem more adequate for the design under consideration. This mechanism allows reducing one of the problems of teaching patterns: the students are not passive recipients of information; now, they are using and doing things in real life. This last issue has a significant and positive impact on learning.

Table 2 Summary of the User's Comments.

	Issue	Description
+	Autonomous and non intrusive	The agent only interrupts the user when it has a suggestion
+	The tool learns from the expert user	With the use of the tool the suggestions are improved
+	The user does not need to know about patterns	It is not necessary to have background knowledge of patterns. The suggestions and the information provided by the agent allow users to learn about patterns and how to use them
−	It does not detect the occurrence of patterns	Sometimes, the novice user applies patterns without knowing this. In this case it could be useful to notify the user, because he can learn that the solution developed includes a pattern
+	It can be consulted about patterns	The user might consult the agent about a specific pattern. In this case, the agent is also used as a source of information about design patterns
−	Novice user evaluations	The agent does not allow the expert user to know how the novice user is using the tool and to evaluate his progress
−	Multiple users	The agent does not work as a distributed collaborative tool
+	Domain independence	The agent monitors the actions of the user when he develops Class and Sequence Diagrams. The problem domain is not taken into account
−	It suggests patterns that may not be necessary	The agent suggests a pattern according to evidence in the user's work. Sometimes, although there are pieces of evidence of a pattern, the user should decide whether he applies it or not depending on the characteristics of the problem

Also, the students learn how to apply patterns in their own problem domain, making it easier to understand the principle behind the patterns.

The current version of *PatternAdvisor* exemplifies the possibility to facilitate teaching patterns, transforming the passive use of a pattern catalogue into an active piece of advice. Now, the user experiments the full lifecycle of a pattern, from problem analysis to implementation.

Perhaps, the main limitation of the approach is that sometimes certain patterns (State or Strategy, for example) are justified if the functionality tends to change into different applications. This kind of knowledge depends on the application requirements and operational environment and should be decided by the developer; students are often inclined to overuse patterns in their designs. Anyway, *PatternAdvisor* can show the potential risks defined in the pattern specification.

Currently we are improving the *PatternAdvisor* so that the user expert can evaluate the learning of the novice users, and how these interact with the agent. We are also working on the development of a collaborative tool, so the teacher and the students can work in the same environment. With the suggested issues (see Table 2) we are improving the agent's detection so that it informs the users when they apply a pattern in their design. We made this because on several occasions the user did not know information about the context and positive and negatives effects of the pattern which was applied.

REFERENCES

[1] S. Sendall, Gauging the quality of examples for teaching design patterns. Workshop on ''Killer Examples'' for Design Patterns and Objects First. OOPSLA 2002, Conference on Object-Oriented Programming Systems, Languages and Applications, Seattle, USA, 2002.

[2] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, Design patterns: Elements of reusable object-oriented software, Addison-Wesley Professional, 1995.

[3] I. Warren, Teaching patterns and software design, Seventh Australasian Computing Education Conference (ACE2005), 2005, pp 39–49.

[4] P. Gestwicki and F. Sun, Design patterns through computer game development, J Educ Resour Comput 8 (2008), 1–22.

[5] J. Biggs, Teaching for quality learning at university, SRHE (Society for Research into Higher Education) & Open University Press, Buckingham UK, 1999.

[6] S. Stuurman and G. Florijn, Experiences with teaching design patterns, In ITiCSE '04: Proceedings of the 9th annual SIGCSE conference on Innovation and technology in computer science education, 2004, pp 151–155.

[7] B. Goldfedder and L. Rising, A training experience with patterns, Commun ACM 39 (1996), 60–64.

[8] P. Company, M. Contero, A. Piquer, N. Aleixos, J. Conesa, and F. Naya, Educational software for teaching drawing-based conceptual design skills, Comput Appl Eng Educ 12 (2004), 257–268.

[9] P. Maes, Agents that reduce work and information overload, Commun ACM 37 (1994), 31–40.

[10] S. Schiaffino, P. Garcia, and A. Amandi, eTeacher: Providing personalized assistance to e-learning students, Comput Educ 51 (2008), 1744–1754.

[11] A. Mitrovic, An intelligent SQL tutor on the web, Int J Artif Intell Educ 13 (2003), 171–195.

[12] T. Heift and D. Nicholson, Web delivery of adaptive and interactive language tutoring, Int J Artif Intell Educ 12 (2001), 310–324.

[13] C. Chen, S. Hsieh, S. Chuang, and S. Lin, A prelab tutoring system for Strength of Materials experiment, Comput Appl Eng Educ 12 (2004), 98–105.

[14] K. Forbes-Riley, D. Litman, and M. Rotaru, Responding to student uncertainty during computer tutoring: A preliminary evaluation, Proceedings of the 9th International Conference on Intelligent Tutoring Systems (ITS), Montreal, Canada, 2008.

[15] M. Evens, S. Brandle, R. Chang, R. Freedman, M. Glass, and Y. Lee, CIRCSIM-Tutor: An intelligent tutoring system using natural language dialogue. Proceedings of the 12th Midwest AI and Cognitive Science Conference (MAICS 2001), 2001, pp 16–23.

[16] P. Suraweera and C. A. Mitrovi, Kermit: A constraint-based tutor for database modeling. Proceedings 6th International Conference on Intelligent Tutoring Systems ITS, 2002.

[17] F. Jensen, Bayesian networks and decision graphs, Springer-Verlag, Secaucus, NJ, 2001, ISBN: 0387952594.

BIOGRAPHIES

Luis Berdun received a PhD degree in Computer Science from the Universidad Nacional del Centro de la Provincia de Buenos Aires, Tandil, Argentina in 2009, the Master in Systems Engineering in 2005 and the Systems Engineer degree from the UNICEN in 2002. Currently he is a Professor at UNICEN where he is member of the ISISTAN research Institute. His research interests include intelligent aided software engineering, planning algorithms, Knowledge Management. More information can be found at http://www.exa.unicen.edu.ar/∼lberdun.

Analia Amandi received a PhD degree in Computer Science from the Universidade Federal do Rio Grande do Sul, Porto Alegre, Brazil, in 1997 and Bachelor's degree in informatics UNL in 1990. Currently she is a Professor at UNICEN where she leads the ISISTAN Research Institute's Knowledge Management Group. She is also a research fellow of the CONICET. Her research interests include personal assistants and Knowledge Management. More information can be found at http://www.exa.unicen.edu.ar/∼amandi.

Marcelo Campo received a PhD degree in Computer Science from the Universidade Federal do Rio Grande do Sul, Porto Alegre, Brazil, in 1997 and the Systems Engineer degree from the UNICEN, in 1988. Currently he is an Associate Professor at UNICEN and Head of the ISISTAN Research Institute. He is also a research fellow of the CONICET. His research interests include intelligent aided software engineering, Software architectures and frameworks, agent technology and software visualization. More information can be found at http://www.exa.unicen.edu.ar/∼mcampo.