

MVCASE Tool – Working with Design Patterns

Daniel Lucrédio ¹
Alexandre Alvaro ¹
Eduardo Santana de Almeida ²
Antonio Francisco do Prado ¹

¹ Computing Department – Federal University of São Carlos
Rod. Washington Luiz, km 235 – São Carlos/SP - Brazil
P.O box 676 – Zip.Code 13565-905 - Phone/Fax: + 55-16-260-8232
{aalvaro, lucredio, prado, trevelin}@dc.ufscar.br

² Informatics Center - Federal University of Pernambuco - Recife Center for Advanced
Studies and Systems
Av. Professor Luiz Freire - Recife/PE – Brasil - University City - Zip. Code: 50740-540
Phone: + 55-81-3271-8430
esa2@cin.ufpe.br

Abstract

Working with design patterns can be considerably improved when using tools that help in pattern creation and application. However, there are several issues involved. For instance, a tool must offer mechanisms to help the Software Engineer to find the right patterns to a particular solution, or to identify and recognize patterns after they were applied. This paper presents the main requirements for these tools, and presents an extension of MVCASE tool to help Software Engineers to create and reuse design patterns in a high abstraction level during the software development process. The tool uses a distributed repository to store the patterns, and uses a code generator to automate most of the implementation process.

Keywords: Design pattern, reuse, pattern application, pattern creation

1. Introduction

Design patterns are among the most important topics inside the Software Engineering research area. According to Florijn et al [10], design patterns describe general solutions for recurring design problems. A pattern¹ is described in terms that explains its usage including, for example, when and how to use it. This description usually follows the pattern format, as can be seen in [11].

¹ Although there are different kinds of patterns, this paper deals only with design patterns. Therefore, the term pattern will be used to reference this kind of pattern.

It is commonly accepted that the use of design patterns in the development of object-oriented software offers several benefits. These include, among others: flexibility and understandability [4], experience encapsulation and reuse, better documentation, and the creation of a common pattern repository. However, as new patterns are created, it may become too difficult, or even impossible, to find one that fits a particular solution, due to the great number of patterns available [1].

In order to avoid this problem and to achieve the benefits, the patterns must be correctly used. For example, the right pattern must be found in order to solve a given problem. Another issue is that the code corresponding to a design pattern must be constructed every time it is applied [6]. It is also possible that two or more patterns are applied together to solve a single problem, which may cause some confusion.

These and other issues may be resolved manually, although it demands some effort. To help in design patterns use, several tools and environments have been proposed. These tools aim to facilitate the creation and application of design patterns. Automatic code generation and pattern structures visualization are examples of some features present in such tools.

In most of cases, these tools deal only with pattern-related issues. However, pattern usage is not isolated from the software development process, and cannot be treated as such. For this reason, in order to be effective, a tool that works with patterns must be integrated with the development process and, if possible, with the tools used for the development.

This paper presents an extension of the MVCASE tool [2,3,5,17,18], which attempts to solve this problem by integrating development and pattern-related activities in a single tool, helping to create and apply design patterns during the development process. The paper is organized as follows: Section 2 presents the main requirements for design pattern creation and application. In section 3, MVCASE tool and its support for design patterns are presented. Section 4 revises the tool in relation to the requirements. Related works are discussed in section 5. Section 6 presents some final considerations and future works.

2. Requirements for Design Pattern Creation and Application

Tool support for working with design patterns involves several issues. Many researches and discussions about this subject can be found in the literature [6,8,10,22]. These works identify mainly two major activities: pattern creation and pattern application. Next, the main requirements for tool support on design patterns are identified, discussing the issues related to these two activities.

2.1. Support for high abstraction level specifications

Design patterns, as the name suggests, represent design-level solutions. These solutions are reflected on the executable code, but do not depend on it. Therefore, a tool intended to support design patterns must be independent of any implementation language. More than that, it is natural to conclude that this tool should work in the same abstraction level as the patterns, in other words, the design level.

2.2. Support for pattern creation

When creating a new pattern, the Software Engineer must be able to create, modify and publish the new pattern, according to the pattern format [11], including all the information required, such as the pattern name, the problem to which it is related, the proposed solution and the consequences. To better illustrate the solution, models, such as

classes or components structures and sample code may be included. Their relationship with other existing patterns may also be included.

2.3. Patterns storage

After creating the pattern, the Software Engineer must have ways to store it in a place from where it can be retrieved for later reuse. The storage mechanism must organize the patterns, storing it together with all the information involved. It is also responsible for:

- *avoiding redundancy*: the same pattern should not be stored twice;
- *maintaining the consistency*: the information concerning the pattern should not be missing or corrupted; and
- *managing patterns references*: in order to improve the user knowledge of the existing patterns and their relations; whenever two stored patterns relate to each other, it should be possible to navigate from one to another.

2.4. Support for patterns searching

Patterns application occurs during the software development process, when a design pattern can be used to help some problem solution. To identify the best suitable pattern for each problem is not an easy task. The software engineer must be aware of the existing patterns in order to choose one that solves the problem. However, if there are too many patterns, it is unlikely that somebody gets to know all of them. Therefore, there must be a searching mechanism which helps the Software Engineer to find a particular pattern, without needing to know all of them.

2.5. Pattern application

After the Software Engineer decided which pattern to use, his chosen solution must be applied into the software that is being developed. The simplest way to do this is to instantiate elements from the pattern, and then bind them to elements in the software. Florijn et al [10] state that three approaches must be considered when instantiating a design pattern:

- *Top-down*: “given a pattern description, generate the necessary design elements in the program and bind them to the pattern. This is typically expressed by: ‘Give me an instance of the Observer pattern’, after which the developer is given an initial set of classes (with methods, relations, etc.) that follows the canonical structure of the Observer pattern”.
- *Bottom-up*: “given a number of elements in the program, bind them to a new pattern instance. This addresses situations such as: ‘These classes (and their methods) together reflect the Proxy pattern; let’s turn it into a Proxy pattern.’ The difference to the top-down approach is that no new design elements are created in the program, instead existing elements are used”.
- *Mixed*: “this approach differs from bottom-up in that the program elements that only partly meet a pattern structure may be combined with newly generated elements that are generated. An example of this is: ‘This structure closely resembles the Composite pattern; turn it into a Composite instance.’ In this case, the structure will be completed with new design elements that were missing”.

However, design patterns application is not just about instantiating some elements and binding them to the pattern. There is a whole solution included, called a *pattern realization plan*, that explains how the pattern is mapped into the language in a particular

circumstance [7]. This plan must be followed in order to correctly apply the pattern. A tool to support pattern application must also have features that help the correct plan execution.

2.6. Pattern tracing

After a pattern is applied, all that is left in the software is a portion of executable code that has no recognizable trace of a design pattern. There must be a way to identify and recognize the patterns that were applied, binding elements of the software to their respective roles in the pattern. In a debate on tool and language support for design patterns, Chambers et al [7] already mentioned the need for this tracing when applying patterns.

2.7. Integration with the development process

As mentioned before, design patterns must not be treated separately from the development process. The same idea is valid for pattern tools, which must interoperate with other development tools. Ideally, a pattern tool should be an integrated part of a software engineering environment, composed by several tools, including modeling and implementation tools, among others. Further discussion about software engineering environments and tools integration can be found in [13, 19, 21].

3. MVCASE tool

In order to better understand how MVCASE supports design patterns, the basic tool features to develop object-oriented software are presented first. Next, its support for design patterns is presented.

The MVCASE (Multiple View CASE) is a modeling tool that provides graphical and textual techniques based on UML [24], to develop object-oriented software. It is fully implemented in the Java language, thus it is platform-independent.

By using UML-based techniques, MVCASE allows the software engineer to specify object-oriented software in different abstraction levels and four different views, as follows:

- *Use Case View*: offers a behavioral view, from the external actors' viewpoint. This view has two main diagrams: the use case diagram, that comprehends a set of use cases, actors and their relationships; and the sequence diagram that details a use case, with a set of objects and a temporal messages sequence [20]. Figure 1 shows use case and sequence diagrams examples, created in MVCASE.

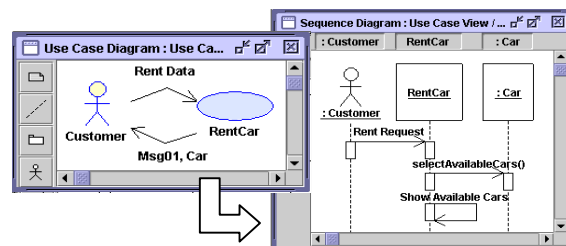


Figure 1. Use Case and Sequence Diagrams.

- *Logical View*: offers a structure of packages, classes and relationships between them. The main diagram in this view is the class diagram, that provides a static view, containing classes, interfaces, collaborations and relationships. Among these relationships are inheritances, associations, aggregations and dependencies [20]. Figure 2 shows a class diagram example in MVCASE.

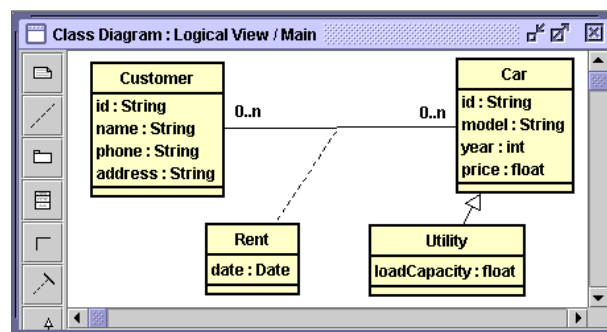


Figure 2. Class diagram.

- *Component View*: offers a static, architectural view, structured in modules. The component diagram is the main diagram of this view. It shows components and their relationships [20]. Figure 3 shows a component diagram example in MVCASE.

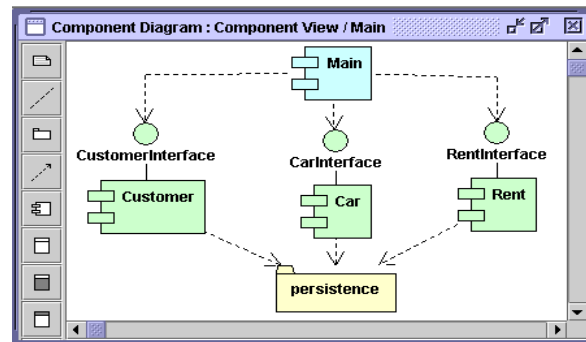


Figure 3. Component Diagram.

- *Deployment View*: offers a static view, showing the deployment architecture. There are nodes and connections showed in a deployment diagram [20]. Figure 4 shows a deployment diagram example constructed in MVCASE.

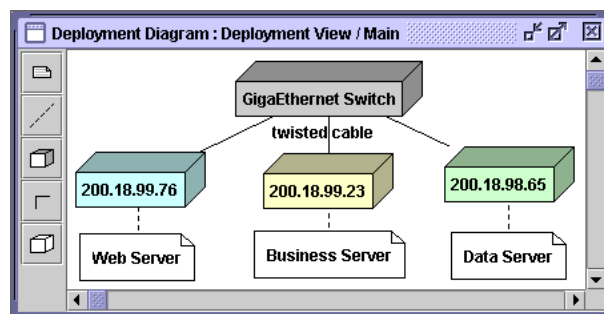


Figure 4. Deployment diagram.

These views are displayed in the tool’s browser, which organizes the packages, elements and diagrams in a tree structure. The browser allows the software engineer to view, create, remove and edit the software elements.

To persist the UML specifications, MVCASE uses the OMG’s XMI standard [25] and a repository. XMI is a XML-based format to represent UML metadata. Based on the four views described above, MVCASE generates XMI descriptions and stores them in the repository. The original XMI format was extended in order to represent graphical information, such as elements position within the diagrams.

The repository can store XMI documents. Since the scope of XMI is not fixed, different kinds of software artifacts can be stored, including UML specifications and executable code, for example. The repository has mechanisms for version control, which means that it can control different versions of a given artifact. It has also security and transaction management mechanisms, to assure the consistency and reliability of the stored artifacts.

The services for storing and searching software artifacts are available through a middleware [16]. By doing so, these services can be accessed over a network. The searching service combines searching and browsing, in order to obtain better results when finding artifacts that are stored in the Repository. Figure 5 shows the repository architecture.

Using the repository, the information produced during the software development process can be stored and managed, becoming available for later reuse. The repository is currently under development, using multi-agents systems technologies and searching engines, being part of a MSc. Dissertation.

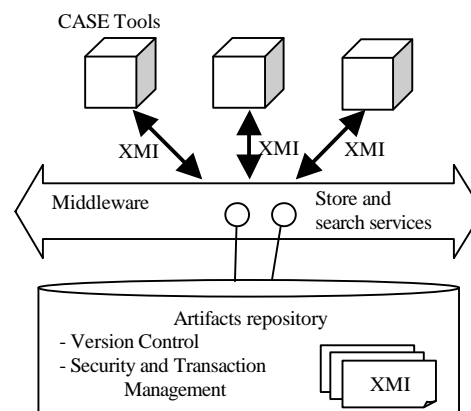


Figure 5. Software artifacts repository architecture.

In order to assist the implementation activities, MVCASE allows the Software Engineer to embed code into the design. Operations can have their behavior described directly in the executable language. This code is then stored together with the high-level descriptions, being used later for code generation. In this way, the Software Engineer can fully implement the software in MVCASE.

MVCASE also provides wizards, to automate tasks such as code generation, packaging and components publication. Until this moment, these wizards support the following technologies: Java 2 [14], EJB [9], CORBA [23] and Java Servlets [15]. MVCASE is currently being used in several Brazilian institutions, serving as a basis for other researches and for teaching.

These are the basic MVCASE features to help the object-oriented software development. Next, its support for design patterns is presented.

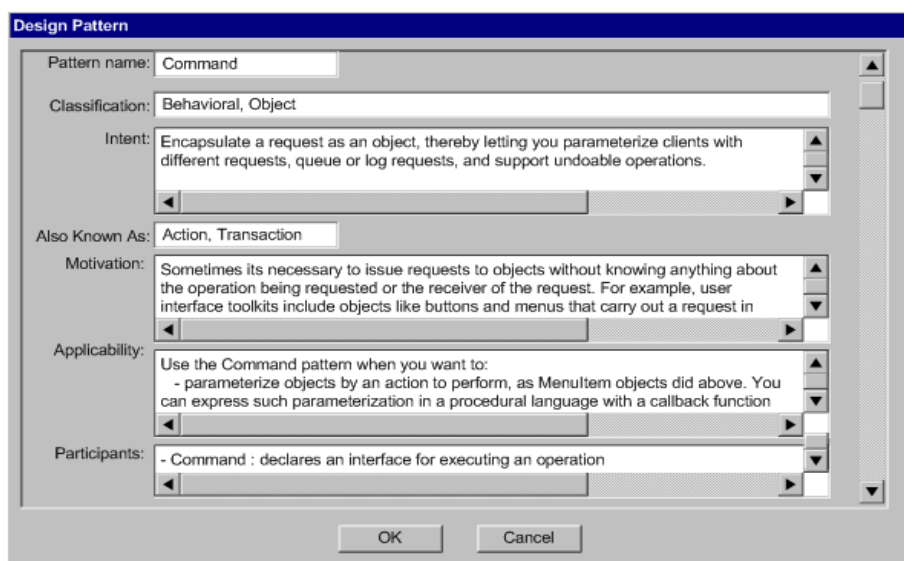
3.1. MVCASE support for design patterns

As mentioned earlier, there are two major activities when working with design patterns: pattern creation and patterns application. Next, a description of how these two activities are performed in MVCASE is presented. Some of these features are not completely implemented yet, such as the repository searching engine. However, the main features are implemented and functional.

3.1.1. Pattern creation

In order to create a design pattern, the Software Engineer must describe it in a way that the problem, the solution, and the consequences related to that pattern can be clearly understood. Gamma et al [11] point out that graphical notations aren't sufficient to describe patterns, and propose a format that is widely used by patterns designers.

With basis on this idea, MVCASE have features to create patterns and describe it either in textual descriptions according to the pattern format [11], and in graphical UML notations. To edit the textual descriptions, MVCASE has an editor which allows the Software Engineer to fill out the several items of the description, as shown in Figure 6, for the *Command* pattern [11].



The screenshot shows a dialog box titled "Design Pattern" with a blue header. It contains several text input fields for describing a pattern. The fields are: "Pattern name:" with the value "Command"; "Classification:" with the value "Behavioral, Object"; "Intent:" with the text "Encapsulate a request as an object, thereby letting you parameterize clients with different requests, queue or log requests, and support undoable operations."; "Also Known As:" with the value "Action, Transaction"; "Motivation:" with the text "Sometimes its necessary to issue requests to objects without knowing anything about the operation being requested or the receiver of the request. For example, user interface toolkits include objects like buttons and menus that carry out a request in"; "Applicability:" with the text "Use the Command pattern when you want to:" followed by a bulleted list: "- parameterize objects by an action to perform, as MenuItem objects did above. You can express such parameterization in a procedural language with a callback function"; and "Participants:" with the text "- Command : declares an interface for executing an operation". At the bottom of the dialog are "OK" and "Cancel" buttons.

Figure 6. Patterns textual descriptions editor.

To create graphical notations, the Software Engineer uses the UML features offered by MVCASE. He can include, in the pattern description, classes and component structures, interaction models, and even executable code. Figures 7 and 8 shows, respectively, the classes structure of the *Command* pattern [11] and the interactions between the pattern objects described with graphical UML notations in MVCASE. Note that the *Execute()* operation, from the *ConcreteCommand*, has some embedded code indicating the invocation of the *Action()* operation of the *Receiver*.

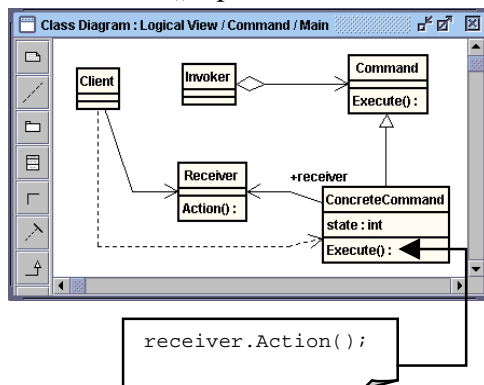


Figure 7. Classes structure and embedded code, of the “Command” pattern.

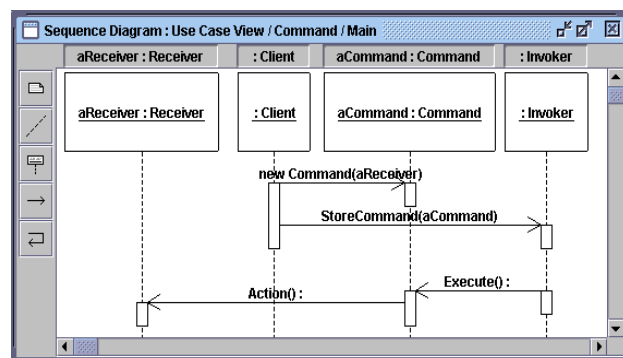


Figure 8. Interactions between the objects of the “Command” pattern, modeled through a sequence diagram.

After the creation of the textual and graphical descriptions, the pattern can be then stored in the repository, from where it can be reused. First, a unique identification is generated for each pattern, aiming to avoid redundancy and to manage pattern relationships. The pattern is then described in XMI, as shows Figure 9, for the *ConcreteCommand* class, of the *Command* pattern. In order to represent pattern-specific information, the XMI format was extended.

Once stored in the repository, the pattern becomes available for reuse through a network. This facilitates its reuse, since different Software Engineers, from different computers, can use the repository at the same time.

3.1.2. Patterns application

In order to support the design patterns application, MVCASE allows the Software Engineer to search the repository for the existing patterns that may help him to solve the problem. The searching is performed in the following way:

1. Initially, the Software Engineer enters some keywords as the searching criteria, aiming to find a pattern based on its name and classification;

```
<?xml version="1.0" encoding="UTF-8"?>
...
<Class ...>
  <Element.name>Command.ConcreteCommand</Element.name>
  <Class.isPublic xmi.value="true"/>
  <Class.superclass>
    <Class xmi.uuid="Command.Command" .../>
  </Class.superclass>
  <Element.contains>
    <Method ...>
      <Element.name>Execute</Element.name>
      <Method.visibility xmi.value="public"/>
      <Method.dimension>0</Method.dimension>
    </Method>
    <Attribute ...>
      <Element.name>state</Element.name>
      <Attribute.visibility xmi.value="public"/>
      <Attribute.type>
        <Class xmi.uuid="int" href="int.xml"/>
      </Attribute.type>
    </Attribute>
  </Element.contains>
</Class>
```

Figure 9. XMI descriptions of class *ConcreteCommand*, from *Command* pattern.

2. Next, the MVCASE repository performs a preliminary searching, comparing the keywords with the names and classification of the stored patterns;
 3. The result of this preliminary searching is then presented to the Software Engineer;
 4. The Software Engineer browses the presented patterns, examining each pattern fields, trying to find one that satisfies his needs; and
 5. Finally, after a pattern is chosen, it is then loaded into MVCASE.
- These five steps are illustrated in Figure 10.

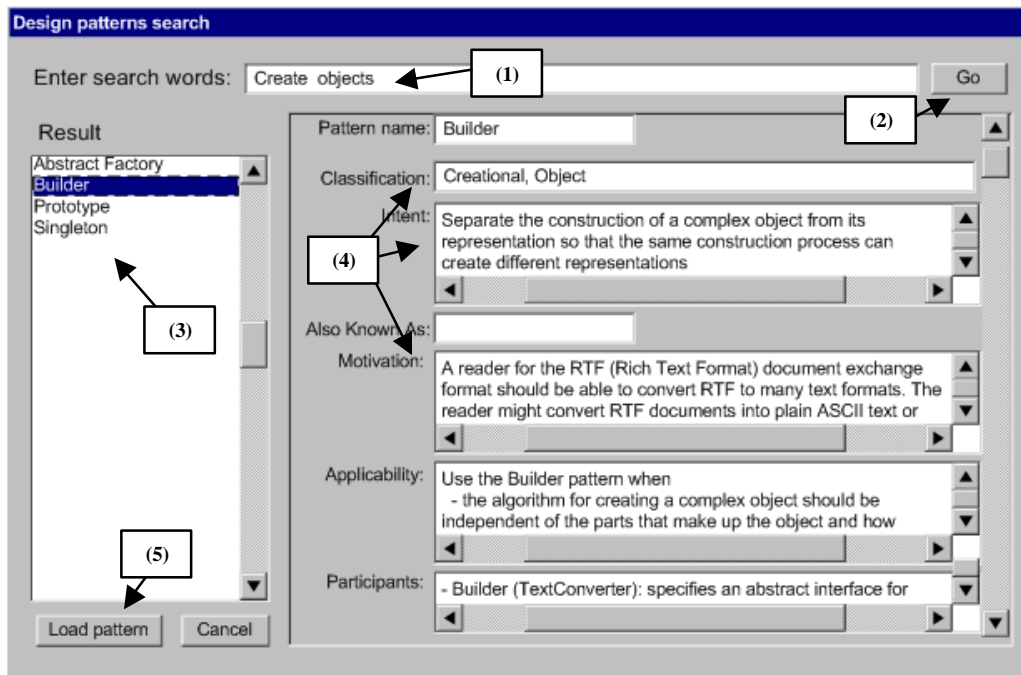


Figure 10. Design patterns search in MVCASE.

The Figure shows the Software Engineer entering the searching keywords “Create objects” (1). Next, he clicks on the *Go* button, which starts the searching engine. The engine is responsible for finding patterns that has names or classification which are similar to the keywords. In this case, *Abstract Factory*, *Builder*, *Prototype* and *Singleton* were found and displayed in a list, since they are classified as “Creational” and “Object” (3). Next, the Software Engineer browses the list, examining the information of each pattern (4). When he finally chooses one, the *Load pattern* button is pressed (5), and the pattern is loaded into MVCASE.

After finding the pattern, the Software Engineer must apply its proposed solution into the software. As seen before, the solution proposed by a design pattern is composed by textual and graphical descriptions. The graphical descriptions are usually high-level models, such as UML models, that shows the solution structure. In most cases, these models serve as a template for applying the pattern, and the Software Engineer replaces the pattern elements with the elements of the software that is being designed.

Since MVCASE works with UML, the graphical pattern descriptions are directly loaded, becoming available in the tool’s browser. Once in the browser, the UML elements can be inserted into the software that is being designed, in a “drag and drop” operation. Figure 11 illustrates this process, where the *Adapter* pattern is being applied.

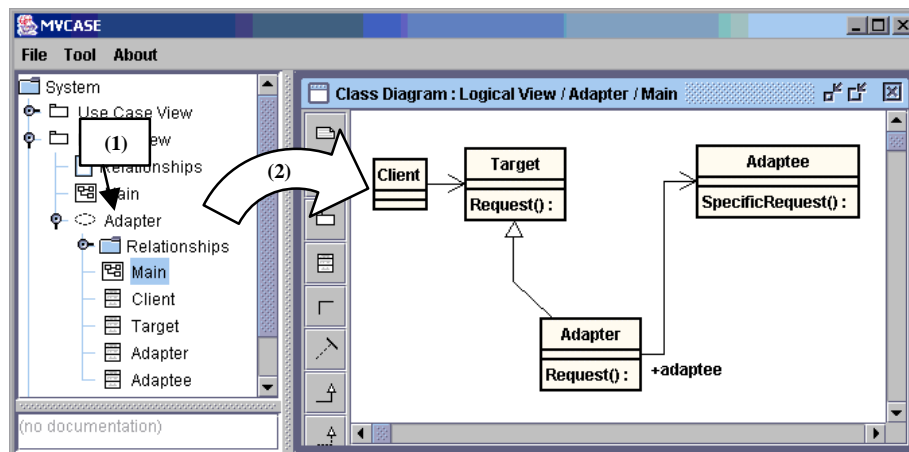


Figure 11. Pattern elements being loaded into MVCASE

In the Figure, the elements of the *Adapter* pattern are loaded into the browser (1). Next, these elements are inserted into the software’s design, in a “drag and drop” operation (2). If there are relationships between the elements, they are automatically inserted. In this particular case, only the class structure of the *Adapter* pattern was inserted into the design, but other UML models could be used as well, such as sequence or component diagrams.

Once inserted in the design, the elements may be edited by the Software Engineer, through the tool features, to complete the software’s design, as shown in Figure 12.

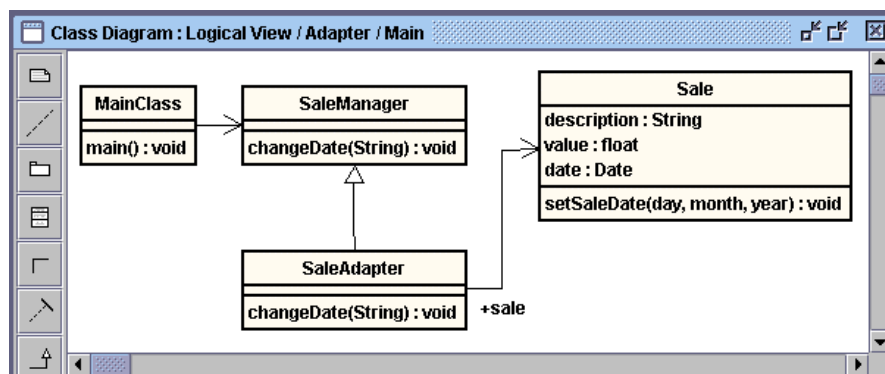


Figure 12. “Adapter” pattern being applied to a system

Figure 12 shows the *Adapter* pattern elements (see Figure 11) edited to become part of the software’s design, in this case, a sales registering system. After the changes, the *Client* class became the *MainClass* class, the *Target* class became the *SaleManager* class, the *Adapter* class became the *SaleAdapter*, and the *Adaptee* class became the *Sale* class.

One important thing that must be stressed is that the code embedded in the operations is still present when the pattern is applied, since MVCASE has features to work with the code in this abstraction level. This code can also be reused, together with the design elements.

In order to indicate that a pattern was applied in some particular design, UML Collaborations are used. According to Rumbaugh et al [20], a Collaboration is a set of elements that work together to offer some behavior that is greater than the sum of its parts. This definition can be applied to design patterns, which are sets of elements that work together to solve some particular problem. Therefore, these abstractions can be used to represent patterns in the design. Figure 13 shows how this is performed in MVCASE.

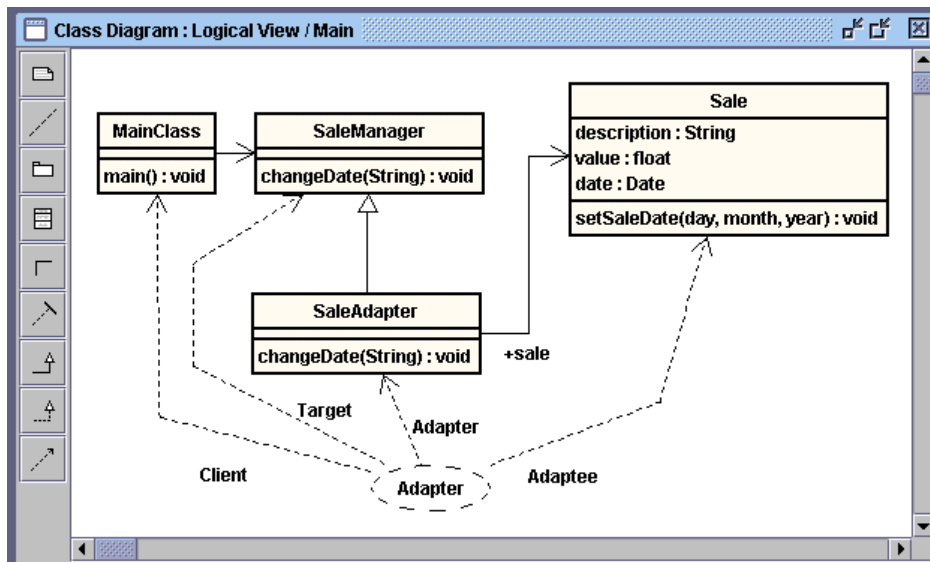


Figure 13. A collaboration to indicate that a pattern was applied

In the Figure, the dashed ellipse represents the pattern. The dashed lines indicate which software elements correspond to the pattern roles. In this case, the *MainClass* class is assigned to the *Client* role. The *SaleManager* plays the *Target* role, the *SaleAdapter* is the *Adapter* and the *Sale* is the *Adaptee*.

By doing this, the Software Engineer can identify in the design which decisions were based on patterns, and which pattern was applied. This helps to document the software, increasing its maintainability.

4. Summary

The previous section presented the main MVCASE features to support design patterns. It supports pattern creation, storage and application, during the software development process. Next a brief discussion about how MVCASE implements each requirement is presented.

4.1. Support for high abstraction level specifications

By providing UML-based techniques to perform analysis, design and implementation activities, MVCASE supports high abstraction level specifications.

4.2. Support for pattern creation

MVCASE supports the pattern creation, allowing the Software Engineer to include textual and graphical information concerning the pattern.

4.3. Patterns storage

The repository supports different kinds of software artifacts storage, in XMI format, including design patterns. To help to avoid the redundancy, an identifier is used to uniquely identify a pattern. However, the management of the relation between the stored patterns is not implemented yet.

4.4. Support for patterns searching

The searching engine uses a preliminary searching with basis on the names and classification of the patterns. Next, with basis on the results of this preliminary searching, it supports the patterns browsing, where the Software Engineer can choose one pattern by examining its information. Although it is very simple, this searching mechanism allows to identify, in a large library, the patterns that may help to solve a particular problem.

4.5. Pattern application

MVCASE is more effective in one of the three approaches for instancing patterns proposed by Florijn et al [10]: the top-down approach. In MVCASE, the elements of the pattern that is being applied are always generated, no matter if they already existed. However, the other two approaches can be used as well, only needing some manual effort to replace the existing software elements with the generated ones.

There is no mechanism implemented to help the correct instantiation of a pattern. Instead, MVCASE relies on the existing structure contained in the pattern's description. For instance, consider the case where two pattern elements must be connected through a generalization relationship. With MVCASE, this generalization is already present when applying the pattern, since its structure is copied into the design of the software that is being developed. Although this does not assure that the pattern will be correctly applied, it facilitates this task.

4.6. Pattern tracing

The pattern tracing is facilitated through the use of UML Collaborations to represent patterns application. With this feature, it is possible to identify and recognize patterns in the final design. The binding between the applied pattern and the code can also be achieved, since MVCASE works in a high abstraction level, treating the code as being an integral part of the design. Therefore, if the pattern can be traced in the design, it can also be traced in the code, since it is embedded in the design. However, although facilitated through MVCASE, the pattern tracing must be performed by the Software Engineer.

4.7. Integration with the development process

In MVCASE, development activities and pattern-related activities are performed at the same time. We believe that this is one of the most valuable contributions of MVCASE as a pattern tool. When developing a software using some tool, the Software Engineer does not want to change to another tool to apply patterns. Also, it is not likely that these tools can be easily integrated, and some manual effort may be needed to make the tools interoperable.

The use of a distributed repository, allowing different users to access the stored patterns simultaneously, also contributes for the integration with the development process, since in most of the cases there is more than one Software Engineer working in the same project.

Although most of the requirements are implemented, some implementations may require refinements and improvements in order to provide full support for design patterns. Some issues, such as the pattern tracing in a high abstraction level, are still being researched by the community, and there is no definitive solution yet. However, we believe that MVCASE offers a practical help for those who want to use patterns in their projects.

5. Related works

Since design patterns were first proposed, several researches attempts to provide tool support for the creation and application of patterns:

Gamma et al [11] present the design patterns concept, and offer clear, practical ways to describe and use them. In our work, we attempt to implement in MVCASE the principles proposed by the authors, such as the format for describing patterns and the way the patterns are selected and used.

A debate on tool and language support for design patterns is presented in [7]. In this paper, the authors discuss the extent to what languages and tools should support design patterns. In particular, the tool support for patterns is discussed. The authors state that the tools still do not fully support design patterns, mainly because the tools are too low-level. They say that a main change is needed: the raising of the conceptual level of tooling. We agree with this point of view, and believe that MVCASE is giving a step forward through this way, by treating the artifacts (particularly the patterns) in a high abstraction level, and embedding the code in the design. This gives the Software Engineer a view of the software that is closer to the “real world” than the traditional programming language view, facilitating, among other things, the use of design patterns.

One of the first attempts to offer tool support for design patterns was proposed by Budinsky et al [6]. The authors present a tool to automatically generate code for design patterns. The code is generated with basis on some specification, and saves the effort of creating the same code every time a pattern is applied. However, as one of the authors of this tool states in a later work [7], this code generation approach has many problems, including:

- although it is configurable, the generated code is not very flexible;
- generated code is often difficult to integrate with existing code; and
- when regenerating some code, important changes may be discarded.

Despite these drawbacks, this first attempt offered many contributions to today’s tools, including our work. In fact, code generation is one of the main features implemented in MVCASE.

Florijn et al [10] proposes another tool to work with design patterns. The authors present some important requirements which represent what is needed in order to offer tool support for design patterns. The proposed tool helps to instantiate new patterns, bind existing elements with patterns, and check if the pattern was correctly applied. Another interesting feature of this tool is that it can be used either in forward engineering and backward engineering. The former is the usual pattern application mode. The latter is performed by identifying pattern occurrences in existing programs and modifying a program to better reflect a pattern structure. Some of these important features are not present in MVCASE, such as pattern application checking and backward engineering techniques. The reason for this is that one of our main objectives was to evaluate the

benefits of integrating development activities with patterns activities, and not providing a complete support for design patterns. However, future works shall deal with these matters.

A discussion about the use of UML to represent design patterns application is presented in [22]. In this paper, the authors discuss the use of UML Collaborations and Parameterized Collaborations to represent design patterns application. Next, they identify many limitations of these abstractions, and present what is needed in order to overcome these limitations. MVCASE uses UML Collaborations to represent patterns application. Given its limitations, this abstraction constitutes a less than perfect solution for representing design patterns in UML, and a more complete treatment is desirable. However, we decided to use UML Collaborations because it would be sufficient to satisfy our pattern tracing requirement.

In [12] a tool to apply design patterns is presented. This tool, called Fred (Framework Editor), uses the concept of *specialization pattern*, which is a specification of a recurring program structure. According to the authors, a specialization pattern is given in terms of *roles*, to be played by (or bound to) structural elements of a program, such as classes or methods. Fred generates descriptions of the tasks to be performed in order to bound program elements to the pattern roles, or it can generate a default form of the element. It also allows the Software Engineer to associate program elements with pattern roles. MVCASE can only generate default specifications, not supporting tasks description or automatic binding between elements and roles. However, Fred only works with executable code, which makes it restrict to the implementation tasks. In MVCASE, instead, patterns are treated in higher abstraction levels, making it suitable for a wider range of development activities.

6. Final considerations and future works

This paper presented an extension of MVCASE tool, to support design patterns usage during the software development process. Some requirements for tool support on patterns were identified, and descriptions of the main MVCASE extensions that implement these requirements were presented.

Several issues remain to be resolved. As seen in previous sections, MVCASE provides efficient support for only one of the three approaches for applying patterns, and a more refined support for the other two should be implemented in future works. Some backward engineering ideas could be applied as well, in order to help in checking patterns application and correcting programs to reflect the patterns structures.

An important issue is the pattern representation using UML Collaborations. As already mentioned, this approach has its limitations, and a better solution is needed, in order to give the Software Engineer an unambiguous way to graphically define and document design patterns.

Currently, only patterns described in a standard format [11] are supported. We decided to adopt this format because it is well accepted and known by most of Software Engineers. However, other formats may be stored in the repository as well, needing only to add specific information in XMI format and to create specific interfaces for searching. Future works may go further in this direction, extending MVCASE's support for patterns.

Other improvements currently under development are related to the mechanisms to store and recover software artifacts (including design patterns) in MVCASE. Multi-agent technologies and more refined searching engines to improve the quality of the repository are being researched.

In this work, the main concern is to evaluate the impact of integrating development activities and pattern support activities into a single tool. Although it has many points that need improvement, the use of MVCASE to apply patterns inside our research group already indicates that this integration offers several benefits, such as the increase of productivity and wider user of the patterns.

7. References

- [1] Agerbo, E., Cornils, A. **How to preserve the benefits of design patterns, In Proceedings of the conference on Object-oriented programming, systems, languages, and applications – OOPSLA'98.** Vancouver, British Columbia, Canadá, 1998.
- [2] Almeida, E., S., Lucrédio, D., Bianchini, C., P., Prado, A., F., Trevelin, L., C. **Ferramenta MVCCase - Uma Ferramenta Integradora de Tecnologias para o Desenvolvimento de Componentes Distribuídos,** XVI Simpósio Brasileiro de Engenharia de Software, Sessão de Ferramentas, Gramado/RS, 2002.
- [3] Almeida, E, S.; Bianchini, C, P.; Prado, A, F.; Trevelin, L, C. **MVCCase: An Integrating Technologies Tool for Distributed Component-Based Software Development.** In: The 6Th Asia - Pacific Network Operations and Management Symposium, Proceedings of IEEE, Poster Session, 2002, Jeju Island – Korea.
- [4] Albin-Amiot, H.; Guéhéneuc, Y.G. **Design Pattern Application: Pure-Generative Approach vs. Conservative-Generative Approach quality.** Proceedings of OOPSLA Workshop on Generative Programming, 2001.
- [5] Barrére, T. S. **CASE com Múltiplas Visões de Requisitos de Software e Implementação Automática em Java – MVCASE,** MSc. Dissertation, Universidade Federal de São Carlos, 1999.
- [6] Budinsky, F.J., Finnie, M.A., Vlissides, J. M., Yu, P. S. **Automatic code generation from design patterns.** IBM Systems Journal, 35(2), 1996.
- [7] Chambers, C., Harrison, W., Vlissides, J.M. **A debate on language and tool support for design patterns.** In Proceedings of the 27th ACM SIGPLAN-SIGACT symposium on Principles of programming languages, Pages: 277 – 289, ACM Press, 2000.
- [8] Conte, A., Freire Jr, J.C., Giraudin, J.P., Hassine, I., Rieu, D. **A tool and a formalism to design and apply patterns.** In Proceedings, SugarLoaf PloP 2002, Rio de Janeiro – RJ, 2002.
- [9] **Enterprise Java Beans (EJB).** Available at site Sun Microsystems, URL: <http://java.sun.com/j2ee/> - Consulted in February, 2003.
- [10] Florijn, G., Meijers, M., Winsen, P. van. **Tool Support for Object-Oriented Patterns.** In Proceedings, ECOOP '97, pages 472-495, Springer-Verlag, 1997.
- [11] Gamma, E., Helm, R., Johnson, R., Vlissides, J.M., **Design Patterns: Elements of Reusable Object-Oriented Software.** Addison-Wesley, Reading, MA, 1995.

- [12] Hammouda I., Koskimies K.: **Generating Pattern-Based Application Development Environment for Enterprise JavaBeans**. In: Proceedings of the 26th Annual International Computer Software and Applications Conference, COMPSAC 2002.
- [13] Harrison, W., Ossher, H., and Tarr, P. **Software Engineering Tools and Environments: A Roadmap**. In The Future of Software Engineering. ACM, New York, 2000, 261-277.
- [14] **Java 2 Platform, Standard Edition**. Available at site Sun Microsystems, URL: <http://java.sun.com/products/j2se/> - Consulted in February, 2003.
- [15] **Java Servlet Technology**. Available at site Sun Microsystems, URL: <http://java.sun.com/products/servlet/> - Consulted in February, 2003.
- [16] Orfali, R., Harkey, D. **Client/Server Programming with Java and CORBA**. John Wiley & Sons, Second Edition, 1998
- [17] Prado, A, F., Lucrédio, D. **Ferramenta MVCASE -Estágio Atual: Especificação, Projeto e Construção de Componentes**, XV Simpósio Brasileiro de Engenharia de Software, Rio de Janeiro/RJ, 2001.
- [18] Prado, A, F., Lucrédio, D. **MVCASE: Ferramenta CASE Orientada a Objetos**, XIV Simpósio Brasileiro de Engenharia de Software, João Pessoa/PB, 2000.
- [19] Pressman, R. S. **Software Engineering: A Practitioner's Approach**, McGraw-Hill 2001.
- [20] Rumbaugh, J., Jacobson, I., Booch, G. **The Unified Modeling Language Reference Manual**, Addison-Wesley, 1999.
- [21] Sommerville, I. **Software Engineering (6th Edition)**. Pearson Education, August 2000.
- [22] Sunye, G., Guennec, A.L., Jezequel, J.M. **Design patterns application in UML**, In Proceedings of the 14 th European conference on Object Oriented programming, Springer LNCS 1850, pages 44-62, 2000.
- [23] **The Common Object Request Broker Architecture: Core Specification, Version 3.0.2**. Available at site Object Management Group, URL: http://www.omg.org/technology/documents/formal/corba_iiop.htm - Consulted in February, 2003.
- [24] **Unified Modeling Language 1.4 specification**. Available at site Object Management Group. URL: <http://www.omg.org/technology/documents/formal/uml.htm>. Consulted in February, 2003.
- [25] **XML Metadata Interchange (XMI) – Version 1.2**. Available at Site OMG. URL: <http://www.omg.org/technology/documents/formal/xmi.htm> - Consulted in February, 2003.