Available online at www.sciencedirect.com
ScienceDirect
Journal of Systems and Software 00 (2015) 1–35
JSS

A Goal-Oriented Approach for Representing and Using Design Patterns

Luca Sabatucci[a], Massimo Cossentino[a], Angelo Susi[b]

[a] *ICAR-CNR, Palermo, Italy*
[b] *Fondazione Bruno Kessler, Trento, Italy*

Abstract

Design patterns are known as proven solutions to recurring design problems. The role of pattern documentation format is to transfer experience thus making pattern employment a viable technique. This research line proposes a goal-oriented pattern documentation that highlights decision-relevant information. The contribution of this paper is twofold. First, it presents a semi-structural visual notation that visualizes context, forces, alternative solutions and consequences in a compact format. Second, it introduces a systematic reuse process, in which the use of goal-oriented patterns aids the practitioner in selecting and customizing design patterns. An empirical study has been conducted the results of which supports the hypothesis that the goal-oriented format provides benefits for the practitioner. The experiment revealed a trend in which solutions better address requirements when the subjects are equipped with the new pattern documentation.

Keywords: Design Patterns, Goal Modeling, Goal Reasoning

1. Introduction

Software patterns are known as proven solutions to recurring problems in the design and the implementation of software systems [1]. This common definition has been refined many times over the years. An interesting definition mentions patterns as instruments for taking decisions during software development [2, 3, 4].

The importance of this observation is that the quality of a software product is highly dependent on the design phase in which strategic decisions are made that remain with the system for the rest of the development. Bad design decisions generally negatively affect the final product [5]. Software patterns help inexpert developers to assess the impact of a decision when the final product is not mature enough to evaluate if a decision is good or not [5].

Since their invention, the response of the research community has been enthusiastic: practitioners have assisted to a phenomenon of proliferation of the categories of patterns, and to an impressive number of collections of patterns

Email addresses: sabatucci@pa.icar.cnr.it (Luca Sabatucci), cossentino@pa.icar.cnr.it (Massimo Cossentino), susi@fbk.eu (Angelo Susi)

addressing a fairly extensive set of problem domains [6, 7, 8]. Patterns exist for solving architectural issues [1], object-oriented design [9], coordination and process problems [10], parallel and concurrency execution [11], security concerns [12] and so on. For instance, the Pattern Almanac [7], published in the year 2000, contains over 1000 patterns. Such a proliferation also generated many duplicates, i.e. patterns that are variants of the same design principle [6]. For instance the Extended Observer [13] and the Middle Observer [14] consider specific application contexts of the original Observer pattern [9].

The value of design patterns is that of being the result of experience on the field gained over several years of trial-and-error attempts. Using a pattern during software development consists in exploiting a well-proven solution, with general benefits on software quality [15] and on maintenance process [16, 17]. Nevertheless the use of patterns in software development practices is far from being trivial. A software pattern is generally less tangible but more flexible than code. A class library provides a collection of classes and methods to use in a black-box fashion. Conversely a pattern is generally described by highlighting a relation between a certain context, a problem, and a solution [1], and it specifies a level of abstraction that is above the level of classes or components. Therefore a pattern is harder to use because its abstractions must first be understood and later be instantiated in specific problem [15].

The most common format for the documentation of patterns is basically text with some visual support and code examples. Advantages are in the richness and flexibility of the natural language and the way it fosters human creativity. Shortcomings are in the potential ambiguity and the average length (in number of pages) necessary for documenting all the details that tend to be spread among many sections of the documentation.

The research community has spent much effort in improving pattern documentation by increasing the level of formalization and the integration with design processes and techniques [18, 19, 2, 3]. Nevertheless most of these approaches are not able to fully represent the abundance of details (for instance they fail to represent alternative and decision points in the solution). In addition, mainly focusing on the solution side, they raise the risk that practitioners confuse a design pattern with its structure diagram [15].

The overall objective of this paper is to propose a novel approach for describing patterns that, on one hand, preserve all the details and, on the other hand, makes accessible the decision-relevant information such as motivation, alternatives, consequences, and forces.

The idea is that, regardless of the category of the pattern, the common element of many pattern descriptions is the *design rationale*, i.e. a set of design goals, design decisions and expected consequences in terms of software qualities. In other words, applying the patterns is similar to making a design decision, which is a cognitive process concerning forces to balance [2, 3] and design decisions to take [20, 21] in order to configure the elements of the system for solving a specific design problem.

Our first contribution is a goal-oriented approach for documenting patterns based on the *i** strategic modeling [22], a conceptual framework for modeling cognitive processes and strategic contexts. The main concepts of *i** are exploited for representing, in a semi-formal notation, a software pattern by preserving all the details that the textual format provides. The *i** notation was built for modeling strategic dependencies in the context of requirements engineering, but

it is general enough to be used in many other organizational application settings. By exploiting this general-purpose nature, the goal-oriented pattern documentation was constructed to be independent of the programming paradigm and the specific category of the pattern. Despite the fact that the notation revealed itself to be expressive enough to document a number of software patterns belonging to different categories existing in literature [23, 14, 24], this paper primarily shows examples of architectural and design patterns taken from the GoF's book [9] or from the Pattern Oriented Software Architecture book series [1]. Only one example of a workflow pattern [10] has been added for discussing independence from the domain.

The second contribution is a systematic process in which the goal-oriented documentation format plays a central role during the software development. The process includes guidelines for a methodical exploration of context problem and forces with the aim of improve the pattern selection activity. In addition the process provides guidelines for driving the practitioner to focus on the design decisions to take for customizing its solution for the specific problem context.

Finally, we report an empirical study conducted to investigate to which extent the goal-oriented format and the related process may offer benefits for practitioner. We empirically observed that class diagrams, created through pattern employment, better meet requirements when inexpert designers are equipped with the goal-oriented documentation format of the patterns. We argue that describing patterns through goals discourages the bad practice of using solutions as code templates; conversely it fosters developers to reason with high level concepts that the pattern embeds and it increases their ability to customize the pattern for the specific problem domain.

This paper is organized as follows. Section 2 analyzes the state of the art and sets up the background for the proposed approach. Section 3 presents the notation used for the goal-oriented pattern documentation. Section 4 illustrates the employment of goal-oriented patterns in a systematic process in which goals aids at discovering which one to use and how to customize it for the specific problem context. Section 5 illustrates an experiment to substantiate our claims. Some remarks about the proposed approach are reported in Section 6 and finally, conclusions are drawn in Section 7. Three complete examples of goal-oriented pattern descriptions are reported in Appendix.

2. Background for the Proposed Approach

The most common format for the documentation of patterns is basically natural language with some visual support and code examples. Depending on the category of pattern the documentation structure slightly changes including a different set of sections. For instance a GoF's pattern is documented through name, intent, motivation, applicability, structure, participants, collaborations, consequences, implementation, sample code, known uses and related patterns. On the other side, a POSA pattern does not explicitly include intent, collaborations and participants, adding a summary, a solution, dynamics and variants. The strengths of such format are due to the natural language: the verbosity and the flexibility allow conveying complex abstractions. This format has been conceived to foster interpretation and creativity. The shortcomings are the potential ambiguity of the natural language, the average length (in number of

pages) and the redundancy of details that are spread among many sections of the documentation.

The design pattern community spent much effort in improving the pattern documentation by raising the level of formalization and the integration with design process and techniques. It is possible to mention declarative [25, 26], formal [18], UML based [19] and semantics approaches [2, 3, 4, 27]. A review is proposed below.

2.1. Formal and Semi-Formal Pattern Documentation

Eden et al. propose LePUS [25], a declarative pattern specification language that uses higher order monadic logic to express pattern solutions. LePUS is based on abstractions of design elements, such as classes, methods, and code and it also includes a visual notation for representing formula of the language. It is strongly based on mathematics and formal logic. They propose a tool, based on Prolog without support for the visual notation. A critic [26] moved to the framework is that, despite the compact form of the visual notation, it often includes too many different syntactic elements making the diagram difficult to interpret. In a successive work Mak et al. [28] propose an extension (ExLePUS) of the initial framework for a better integration with CASE tools also discussing the problem of compound patterns. Patterns contain slots that are filled by other patterns to produce an interconnected architecture.

Mikkonen proposes DisCo [18], (Distributed Co-operation) which uses a form of Temporal Logic of Actions to formally describe constraint interactions for reactive systems. Therefore, while LePUS focuses on the static aspects of patterns, DisCo is primarily concerned with behavioral aspects. The framework allows for managing interactions among objects whose correctness is ensured by property-preserving refinements.

Both LePUS and DisCo (and their available extensions) greatly reduce the ambiguity of the pattern solution, but they must be complemented with the traditional documentation for what concerns the other aspects of pattern description. Another note is that the proposed level of formalization requires special skills to interpret formula.

In addition, many semi-formal approaches exist in literature, most of which are based on UML [29, 30, 31, 32, 33] so to be easily integrated into design practices. However these approaches inherit some limitations from the UML: 1) they are not able to represent alternatives and decisions points; 2) they focus on static and dynamic diagrams enhance the common misconception of confusing a design pattern with its structure diagram [15]. The risk is that of downgrading pattern usage to a mere design/code template. Practical experience teaches that this error is frequent [15]: in particular the UML diagram often shown as illustrative structure of the pattern is often considered as the pattern. Conversely, it is generally an example of one of the forms the pattern may take [34].

A Design Pattern Modelling Language (DPML) [26] has been proposed to model and use design patterns according to separation of concerns between design patterns, design pattern instances and object models. An ad-hoc visual notation is proposed for aiding to manage patterns. This may help in describing pattern structures in terms of participating objects and relations between those objects. In particular the novelty of the notation is the concept of dimension, a construct used to indicate that each participant represents a set of objects in the object model, instead that just a single object.

Zdun [2] proposes to define a common grammar for pattern languages in order to simplify the pattern selection process from a catalogue or from many catalogues. This semantically based pattern language grammar may be derived from the design solution space. This work makes explicit positive/negative contributions of a pattern towards quality factors.

In [3], authors make context forces explicit as non-functional requirements (NFR). They also introduce the concept of design goals, leading the designer to explore many alternative impacts of each pattern over these non-functional requirements. Weiss [4] carried out an extension of this approach by introducing a rigid form for goal hierarchy to reason about a pattern. This structure is built as the interaction of NFRs coming from the pattern with NFRs coming from the problem context.

The latter three approaches [2, 3, 4] support the idea that a pattern is an instrument for taking decisions. The main objective is to facilitate the pattern selection activity by proposing elements for making the best solution according to the characteristics of the problem. The level of granularity specified for the goal is at pattern level: each pattern is associated to one or more goals. To the best of our knowledge, there is not a pattern description technique that introduces a way to manage alternative solutions of a design pattern structure for the customization of the solution for the application context.

2.2. Three Dimensions of a Design Pattern

The process suggested in the GoF's book [9] and refined in many successive works [29, 30, 31, 35, 33] proposes three fundamental steps.

1. To identify if conditions hold for which a pattern could be useful: i.e. the problem matches with the pattern purpose. Since the pattern is created to be as general as possible, the practitioner has to pay attention on applicability and consequences of a specific pattern to be sure that it is the right one for solving a specific problem.
2. To identify classes of the system that the pattern will affect, or introduce new classes when necessary. The description of responsibilities and collaborations associated with each participant may be used as a guide.
3. To modify involved classes according to the solution by introducing all the necessary elements (interfaces, abstract class associations, and so on). Implementing the responsibilities and collaborations in the pattern may complete classes. The practitioner has to pay attention on the implementing issue section in which several alternative solutions are described; each alternative solution provides a different balance to system forces.

From these specific object-oriented guidelines it is possible to abstract three conceptual dimensions: let us refer them as the WHY/WHEN, WHO and HOW dimensions.

The WHY/WHEN dimension matches with the pattern purpose and it answers the question 'which conditions must hold in order to consider a pattern useful'. This conceptualization raises from observing that *design goals* are central elements of design activity, and a design pattern is an instrument to address some of these goals. A pattern can

be selected if it is able to address a design goal that matches with the current problem design goal (WHY). However also the identification of system forces drives the selection of a pattern. These may be *desired properties* the system should have, or *side effects* coming from the interaction with existing elements of the system. A pattern can be selected when its consequences match with the current design context (WHEN).

The pattern solution orchestrates participants (sometimes mentioned as placeholders or roles). In an object-oriented solution these are classes and objects, whereas in a architectural context these are components of the system. The WHO dimension deals with the binding between participants and responsibilities; it answers the question 'which elements of the system should participate to the solution and which responsibility are assigned to them'. When customizing a design pattern for a specific context problem, it is necessary to reason on the 'WHO' dimension. The first step is to identify which elements of the system participates to the structure suggested by the pattern. Consequently these elements are charged of (individual or collective) responsibilities prescribed by the pattern solution. When requested, the practitioner has to select, among more alternatives, the most appropriate for the problem context.

Finally applying a pattern solution means to adopt a specific design model, thus introducing the effects of the decisions made in the previous steps. The HOW dimension embeds this activity by exploring the question 'in which way the problem context is addressed by a specific design model'. This conceptualization derives from the idea that the solution of a pattern prescribes a recipe (including possible variants) for addressing the design objective. The practitioner, by following the instructions of this recipe, can implement the solution without being an expert in solving that specific problem. In most pattern documentation this part is generally facilitated by visual aids. For instance class diagrams and interaction diagrams are typically employed in the GoF's book. It is worth stressing that (i) implementing a solution is a customization activity that transfers a general idea to a specific context, where each instruction is justified by a specific purpose and (ii) when many variants exist for a pattern, each decision made in the previous steps allows a finer customization of the solution.

Figure 1: The three dimensions of the pattern description.

2.3. From Goal-Oriented Concepts to Patterns

The design is conceived as the complex human activity of solving software design problems in which each task is triggered by a design goal: for instance an architectural objective, a specific feature to implement, a quality the system must have and so on. Each of these identifies a potential opportunity to take decisions that will influence many aspects of the system-to-be [36].

Examples of decisions to front during the object-oriented design activity are: to determine the object granularity, to choose inheritance versus composition, to implement sub-classing versus delegation.

If the design is the activity of solving software design problems, patterns are the means for recording cases of success in which a design task have been properly solved, also documenting why this happened and in which cir-

cumstances the same strategy can be reused. The proposed goal-oriented format is intended to facilitate the access to relevant information that drives a practitioner to take the correct decision: motivations, design alternatives, consequence and forces.

The proposed approach for documenting patterns is based on *i** that is Eric Yu's seminal work, a conceptual framework for modeling cognitive processes and strategic contexts. The main concepts of *i** are exploited for representing, in a semi-formal notation, a software pattern by preserving all the decision-relevant information the textual format provides. In *i** , goals are first-class citizens of a graphical language with the intrinsic capability to express alternate visions of the desired outcome [22].

Goals are *intentional entities* because they are used for associating an actor's behavior to a motivation. For instance the sentence "I play the lottery because I want to become rich" embeds an intention (to become rich) and a way to pursuit it (playing the lottery). The *i** notation allows to model situation like that by employing concepts such as actor, goal, task, contribution and decomposition. The goal-oriented pattern documentation uses *i** concepts, specializing them in a design context (design goal, design task, etc.). The elements of the notation for a goal-oriented pattern are illustrated below.

- A *Design Goal* is a desired design outcome that the designer wants to meet. A goal may derive from (i) the need to solve some design problems emerging during the development of a system or (ii) the need to ensure non-functional requirement or quality assets. The design goal is enough abstract to be used to express several categories of design problems; for instance [to make user-interface easily extendable] is a design goal for the MVC (architectural pattern), [to define a one-to-many dependency] is a design goal for the Observer (behavioral pattern) and finally [to separate the construction of an object from its representation] is a design goal of the Builder (creational pattern).

- An *Actor* is an entity directed toward some object or state of affairs (design goals). This abstraction allows explicitly representing who is responsible of addressing design goals. Each 'Pattern' is an actor that owns a set of high-level goals. In addition, each 'Participants' of the pattern is represented as another actor that owns sub-goals, i.e. goals obtained as decomposition of high-level goals. An example of pattern actors are the Mediator (pattern actor) who desires [to implement many-to-many relationships], the Colleague and the Mediator (participant actors) in charge respectively of [to manage behavior parts] and [to manage the control] (see Figure 2).

- A *Design Task* specifies a concrete way of addressing a design goal. It represents a step for implementing the whole pattern solution. An example of design task is [make all the colleague classes inherit from the abstract_colleague class]. At the end of the customization activity the pattern solution is described as a sequence of design tasks the practitioner have to execute.

- Finally, A *Design Resource* is an abstraction for indicating entities of the design model. Resources are mentioned inside design goals and tasks. Design goals express some kind of quality assets the resource must have

(for instance [to avoid subclass proliferation]). On the other side design tasks express instructions for manipulating resources in order to address some design goals (for instance [add a public *request* method]). When working with object-oriented patterns the resources will be classes, attributes, methods, interfaces and so on. Conversely, when working with architectural patterns the resources will be components, interfaces and responsibilities. This is an elegant way to make the notation independent of a specific category of pattern and to make elements of a particular programming paradigm interchangeable.

3. The Goal-Oriented Pattern Description

The goal-oriented pattern description spans across three views which details different aspects of the same pattern[1]. Hereinafter we use the term ***3V-Pattern***, and the abbreviation 3VP, for indicating the conjoined use of these three views for providing a design pattern.

- The first view is the *Strategic Model* that concerns pattern's motivation and applicability. It summarizes main design goals the pattern address and the distribution of responsibilities among pattern actors. It is 'strategic' because it allows the practitioner to perform a preliminary evaluation on the usefulness of the pattern for a specific problem.
- The second view is the *Design Goal Model* that provides details about how the main goals are decomposed into sub-goals and tasks. It also considers alternative ways to address the main goal and how each alternative solution impacts the system forces.
- The third view is the *Design Scenario* that concerns with pattern implementation. It describes in details each design task mentioned in the goal-model. It also suggests the correct sequence of execution in order to modify the existing system and to implement the solution.

In the remainder of this section the three views are described in details.

3.1. The Strategic Model

The *Strategic Model* depicts the intentional structure of a pattern by considering a design pattern as model for a social organization in which elements of a system are potential participants. This view also details relationships among participants and the responsibilities for each element of the system that will collaborate.

The corresponding diagram (see Figure 2) provides a higher-level characterization of a pattern through actors, design goals and dependencies. The view includes the pattern and all its participants that are represented as actors. An example of strategic model for the *Mediator* pattern is shown in Figure 2. The proposed notation represents the

[1]The definition of View is provided by the ISO/IEC/IEEE 42010 standard, available at `http://www.iso-architecture.org/42010/`

design pattern and all its roles as actors (circles). The pattern actor is attached to rounded rectangles enclosing natural language text: these are design goals the pattern address, i.e. these are the main motivation for reusing the pattern.

The actors in a strategic model are connected in a network of dependency-relationships that illustrates the rationale behind the distribution of responsibilities. A *Dependency* describes how a source actor (the depender) depends on a destination actor (the dependee) for an intentional element (the dependum). The dependum is generally expressed through a design goal, thus to specify the nature of the dependency and its motivation. However two actors may also depend for a resource. The Dependency relationship is represented as a sequence arrow-goal-arrow; it delegates a responsibility (goal) from a role to another role. In Figure 2 the colleague role is delegated to [manage behavior parts] whereas the mediator role is delegated to [manage the control].

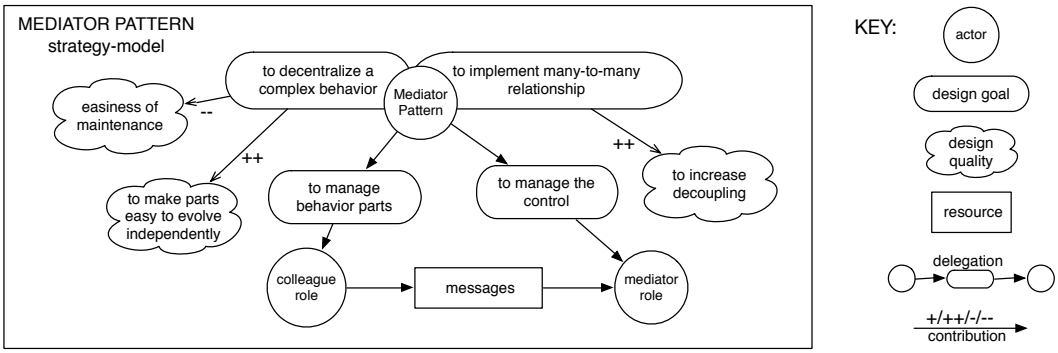


Figure 2: Example of strategic-model for the *Mediator*.

Providing details about the quality assets of the pattern completes the strategic model. These are also design goals but they are represented with a different graphical notation (clouds) because they are qualitative assets (not directly measurable).

The *Contribution* is a further relationship to be used in the modeling for connecting two intentional elements [22] — goals, tasks and resources — whose precise semantics is defined in [37]. In our work, contribution links may be used to connect Goals to Goals, Design Tasks to Goals, and even Goals to Design Tasks. The relationships is represented as an arrow with one of the +/++/-/ annotations that represent a qualitative measure of the contribution. A (strong) positive contribution is the measure of benefits that, for example, the achievement of a goal provides to another goal. Similarly, a (strong) negative contribution is the measure of drawbacks that the achievement of a goal provides to another goal. Contributions are useful for the balance of trade-offs during pattern selection and customization. For example, Figure 2 indicated that [to decentralize a complex behavior] strongly contributes [to make parts easy to evolve independently] but globally it has very negative impacts on the [easiness of maintenance].

3.2. The Design-Goal Model

The core view of the pattern description is the *Design Goal Model* that depicts the strategic rationale of a pattern. This view represents the decision-making process that drives a practitioner to move from a generic design goal to a

concrete pattern solution. Figure 3 is an example of design-goal-model for the *Mediator* pattern. In order to simplify the graphical notation and improving readability when the diagram becomes dense, goals are represented as minimal text. The root goal of the tree represents the main design goal the pattern wants to achieve. It must correspond to main goals represented in the strategic model. For the mediator pattern the root goal is the composition of two design goals: [to implement a many-to-many relationship] and [to decentralize a complex behavior].

The main instrument of this view is the *Decomposition* relationship that allows refining goals or tasks into sub-goals or sub-tasks, thus generating a hierarchy of intentional elements. Two strategies of decomposition are possible: 1) the *AND decomposition* prescribes that the satisfaction of all the sub-goals are necessary for the target goal to be satisfied; 2) the *OR decomposition* prescribes that the satisfaction the target goal is delegated to the satisfaction of one of its sub-goals.

The AND decomposition supports the practitioner with an argumentative style of reasoning: by following the chain of decomposition it is possible to understand the motivation behind any choice of the pattern. The main strength of goal models is the natural capability to represent alternative paths for satisfying the main goal. In fact, the OR decomposition provides a description of mutually exclusive ways of satisfying a design goal. In other words each OR decomposition introduces a decision point for the practitioner.

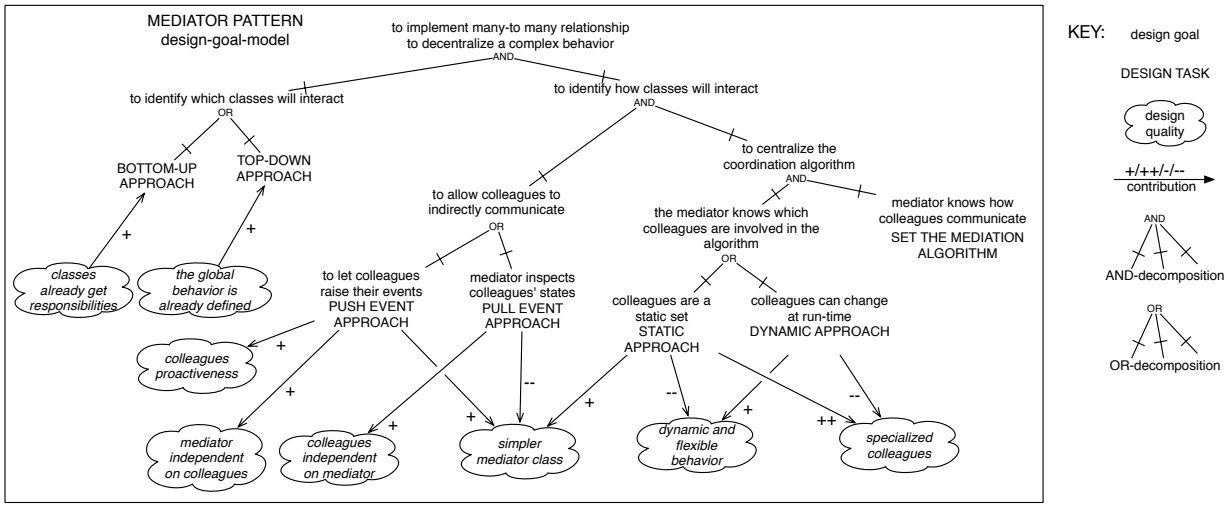


Figure 3: Example of design-goal-model for the *Mediator*.

In Figure 3 the root goal is AND-decomposed into two sub-goals: [to identify which classes interact] and [to identify how classes interact]. At the same way, [to identify how classes interact] is decomposed into [to allow colleagues to indirectly communicate] and [to centralize the coordination algorithm]. Also the goal [to centralize the coordination algorithm] is decomposed into [mediator knows which colleagues are involved in communication] and [mediator knows how colleague communicate].

Conversely, the OR decomposition introduces a Decision Point for the practitioner, i.e. a decision about which variant to choose for the achievement of a design goal. This instrument supports the designer with the necessary

information for taking the best decision according to the current problem context. Indeed, each alternative path may be analyzed by looking at positive/negative contributions towards some quality assets. This enforces the practitioner to reason on possible consequences into the system. An example of decision point is the goal [to allow colleagues to indirectly communicate] that is achieved either by [to let colleagues to raise their events] (marked as [PUSH EVENT APPROACH]) or by [mediator inspects colleagues' states] (marked as [PULL EVENT APPROACH]). The designer, according to the specific design context, may choose between these two candidate solutions considering that the [PUSH EVENT APPROACH] fosters [colleagues proactiveness], [mediator independent from colleagues] and [simpler mediator class], whereas the [PULL EVENT APPROACH] is positive for [colleagues independent from mediator], but is strongly negative for [simpler mediator class].

Finally, leaf nodes in the goal hierarchy are Design Tasks, i.e. blocks of instructions for implementing the solution. In the goal-diagram, design tasks are represented in capital letters. For instance the [PUSH EVENT APPROACH] is the design task directly associated to a goal. Sometimes a goal may be addressed through many tasks. Introducing the corresponding decomposition relation represents this. For example, the design goal [to identify which classes interact] can be achieved by two alternative design tasks: [BOTTOM-UP APPROACH] or [TOP-DOWN APPROACH].

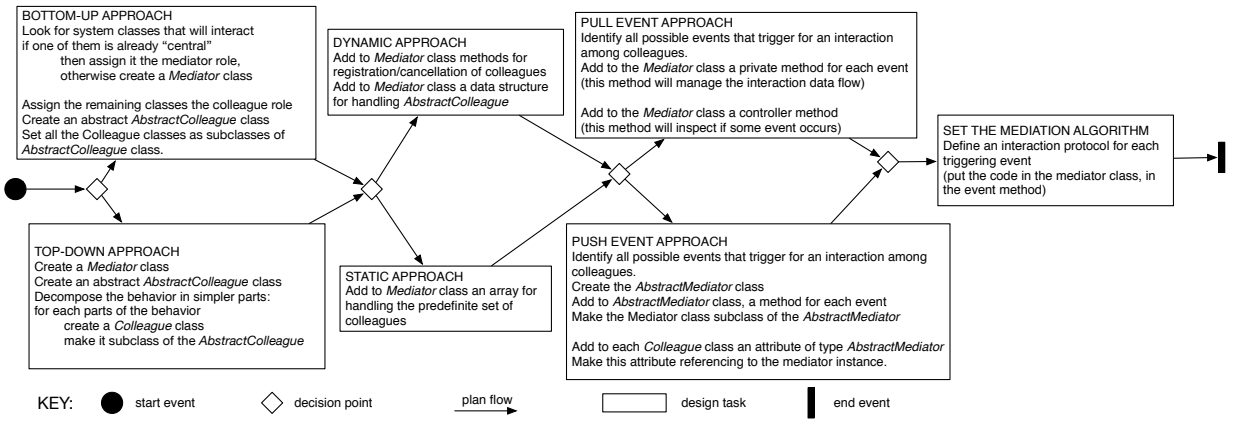


Figure 4: Example of design-scenario for the *Mediator*.

3.3. The Design Scenario

The *Design-Scenario* view enters in the details of each design task by providing precise instructions for the practitioner. An example is shown in Figure 4. It is a dynamic view similar to a flow of activities. The black circle represents the starting point, whereas the vertical bar is the ending point. Each diamond corresponds to a decision point encountered in the design-goal-model view.

Each rectangle is a design task that explores in details the set of instructions necessary for addressing a goal or a sub-goal. Inside the rectangle there are design instructions provided as informal sentences. an example of instruction from Figure 4 is [create an AbstractColleague class].

When executing a design scenario, many possible object models can derive. Each design task generates a portion of the whole solution, thus, any design decision introduces alternative solutions. When a scenario contains more decision points, then the number of solutions grows as a result of their combination.

4. The Process for Selecting and Customizing a Goal-Oriented Pattern

The use of a pattern is strictly correlated to taking decisions and selecting variants that fit with a specific problem context [21]. The proposed process offers a systematic approach for pattern selection and customization in which the three dimensions WHY/WHEN - WHO - HOW are explicitly related to design activities (Figure 5). The process deals with two different reuse scenarios: (i) generative: pattern employment generates new design resources (classes/objects as well as components/interfaces) and (ii) refactoring: the system already exists and the practitioner has to change properties of the model by employing a pattern.

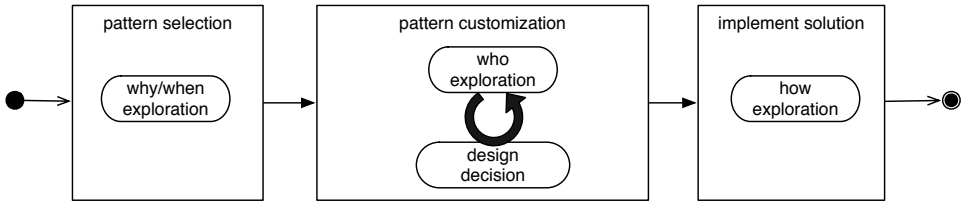


Figure 5: Phases and main objectives of the activities for implementing a pattern solution.

4.1. Pattern Selection Phase

The identification of design needs triggers the pattern selection phase dealing with the well known *the pattern decision problem* [20]. A design need is mapped to a design goal that the designer manages to address by using a pattern (Section 3). The phase helps the practitioner at discovering which pattern is suitable for the given design goal, also considering non-functional requirements and forces to balance.

When many candidate patterns are evaluated for addressing the same problem, the pattern strategic-model may simplify the balance of *context forces*. Indeed this view highlights relevant features that make patterns easily comparable.

In the example of the Mediator pattern (see Figure 6), the main goals [to implement a many-to-many relationship] and [to decentralize a complex behavior] come together with positive or negative contributions towards quality aspects. In particular, in the case of the Mediator pattern quality aspects are [to increase decoupling], [easiness of maintenance] and [to make parts easy to evolve independently].

In particular the **WHY/WHEN Exploration** activity represents a systematic comparison of the couple ⟨context problem , system forces⟩ with the couple ⟨design goal , quality aspects⟩ associated to the pattern.

When the main goal matches designer's needs and the balance of the forces is positive, then the pattern is a good candidate for solving the design problem. The required matching is a *semantic* matching, that is more flexible than

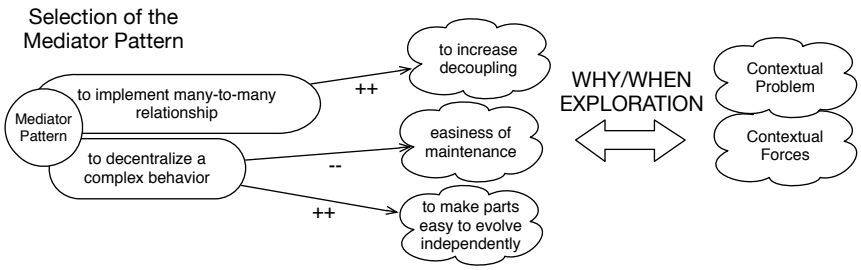


Figure 6: Example of contributions for the selection of the Mediator pattern.

syntax matching because it involves the meaning the expressions in natural language. Clearly, human interpretation is fundamental for taking decisions during this activity.

For instance the sentence [to make parts easy to evolve independently] semantically matches with the sentence [to allow future extensions of a family of classes].

4.2. Pattern Customization Phase

When a pattern is selected, then the designer is in charge to refine the proposed solution, according to the specific problem context. This phase is full of decisions to take.

The first choice concerns how the elements of the system will participate to the proposed solution. Indeed almost all pattern solutions are organized as a set of interacting roles [38]. Instantiating the pattern solution means to select which of system elements will play these roles. The aim of **WHO Exploration** activity is to support this decision by the strategic-model. It provides an explicit semantic of responsibilities that are associated to each role. The advantage is to increase the designer awareness about the consequences of assigning a role to a system element (existing or to-be).

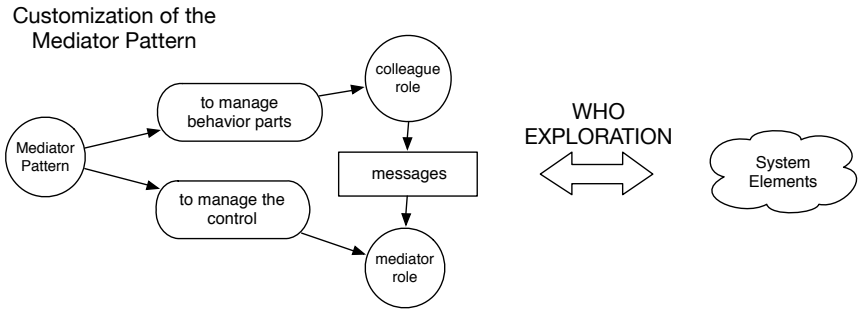


Figure 7: Example of guideline for identifying and assigning pattern actors to system elements.

In the example of the Mediator pattern, the strategic-model (Figure 7) indicates that members of the colleague role will be responsible [to manage behavior parts], whereas members of the mediator role will be responsible [to manage the control]. In addition the colleague role will deliver [messages] to the mediator role.

The second step, in order to customize the pattern solution, is the **Design Decision** activity, related to evaluating and deciding among all the alternatives contained in the design-goal-model. Decision points are located in OR decomposition relationships of the view. In order to lead the designer to the best choice, every alternative path is associated to positive and negative consequences (qualities of the design). By giving a preference to the proposed quality aspects the result is a force balance.

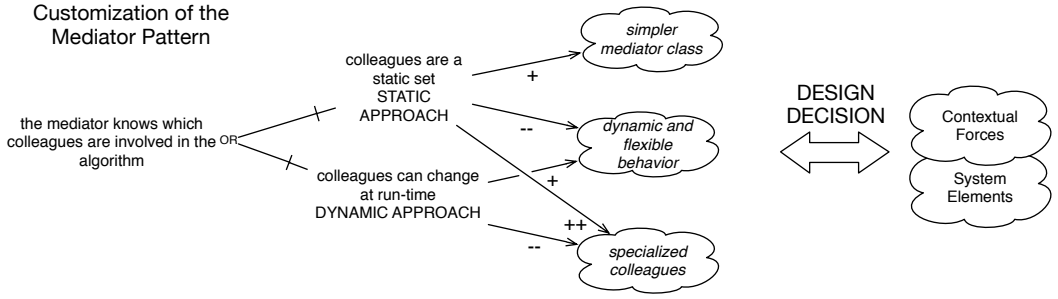


Figure 8: Example of support to decisions encapsulated into the Mediator pattern solution.

An example of design decision is reported in Figure 8. In the Mediator pattern, one of the decisions is related to the goal [the mediator knows which colleagues are involved in communication] and two alternatives are proposed: 1) [colleagues are a static set], or 2) [colleagues can change at run-time]. The first choice has the consequence to generate a [simpler mediator class], but it has a negative impact in creating a [dynamic and flexible behavior] because the set of colleague cannot change at run-time. On the other hand, the second choice is preferred when creating a [dynamic and flexible behavior], even if it is a negative impact in implementing a [specialized behavior], because all of them will have the same interface. If the designer requires a dynamic behavior then it is trivial to decide what option to select. Other cases exist in which the force balance is quite more complex, since there are many positive/negative contributions versus many quality aspects.

4.3. Solution Implementation Phase

This phase aims at implementing the pattern solution, i.e. to apply the solution to the model of the system. The design-scenario view aids the designer to modify the system under-development in order to implement the solution. First the design-scenario view is refined on the base of the decisions taken in the previous phase. When all the decision points are resolved, the result is a straightforward sequence of design tasks to execute.

The left side of Figure 9 shows an example of refined scenario in which the designer selected (i) the bottom-up approach, (ii) the static approach and the (iii) the pull event approach. On the right side of the figure, the resulting class diagram is shown. The numeric annotations in Figure 9 represent the connection between design tasks and their impact to the resulting class diagram.

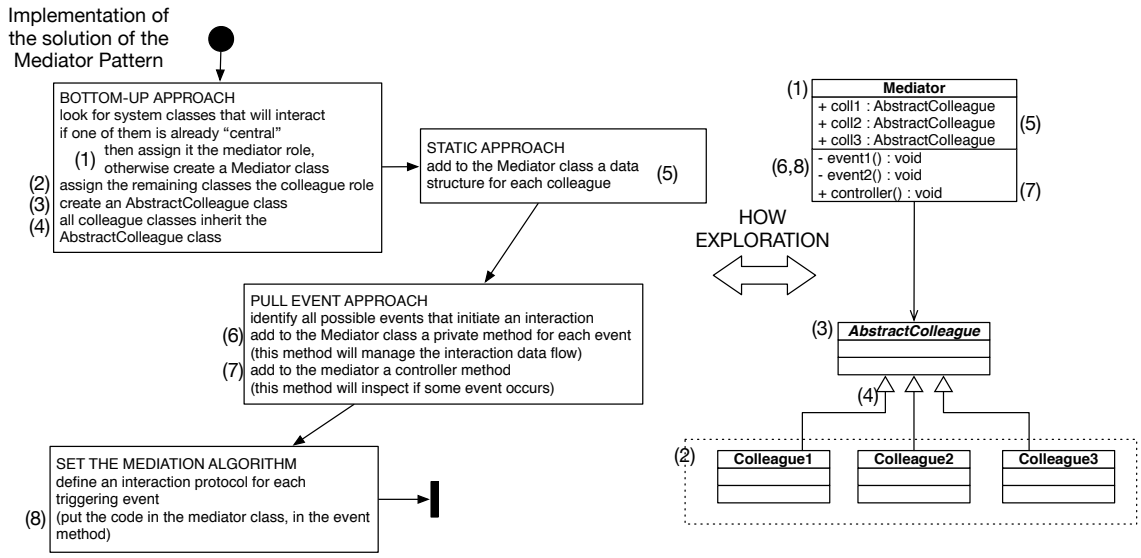


Figure 9: Example of guideline for implementing the Mediator solution to the system.

5. An Experiment for the Evaluation of the Approach

This work has been conducted under the intuition that the pattern documentation format has an impact on the reuse process. This section presents a controlled experiment, designed according to the guidelines of Wohlin et al. [39], for observing how the use of a different documentation format results in different outcomes in terms of quality of the produced model. This experiment represents the opportunity of validating of the 3V-Pattern notation in the context of object-oriented (GoF) patterns. However the validation of the 'reuse process' is out of the scope of the empirical study, and it is left as future work.

5.1. Experiment Design and Execution

The goal of the experiment is *to analyze the pattern reuse process for the purpose of comparing the impact of the documentation format into the design activity*. The quality focus regards the *quality of the pattern customization when a specific documentation format is provided to unskilled pattern users*. The *context* of the experiment is composed of *subjects* (students and young researchers with few experience in using patterns) and *objects* (the design tasks to solve by the use of assigned patterns). Table 1 summarizes the major characteristics of the experiment.[2]

5.1.1. Subjects

The experiment consists of a study with 12 computer science students and young researchers of the University of Palermo, in Italy. Such subjects have already attended Software Engineering and Object-Oriented programming

[2] All the material of the experiment is available at `pa.icar.cnr.it/aose/patterns/Experiment.zip`, including the experiment design, the description of tasks, and the results.

Table 1: Overview of the experiment.

Goal	To compare the impact of two different pattern documentation formats (Gamma-like documentation — GoF — and Goal-based pattern description — 3VP) on the quality of the produced models.
Context	Two different design tasks (*BankLoanSystem* and *RoboCup*) to be performed using the two different pattern documentation formats (Gamma-like documentation — GoF — and Goal-based pattern description — 3VP); 12 computer science students and young researchers unskilled in pattern employment.
Main Null Hypotheses	the goal-oriented pattern description does not increase the quality of the produced model.
Independent variables	*Exercise*, *Pattern documentation format*.
Other factors	*Order of the labs*, *System* (Application), *Background*, *OO experience*, *UML experience*, *Design Pattern experience*.
Dependent variables	*Correctness*.

courses and they are fairly skilled in Java programming. They also had familiarity with UML modeling and scholar knowledge about design patterns but no or very limited practical experience in employing patterns. Some subjects also hold industrial experience as part-time software developers.

5.1.2. Hypotheses Formulation

From this experiment we expect that 3VP increases the quality of the outcome with respect to the employment of traditional GoF.

Based on this informal statement, we can formulate the subsequent null hypothesis H_0 and alternative hypothesis H_a to be tested:

> H_0: goal-oriented pattern description does not increase the quality of the produced model, $H_0 : quality(GoF) \geq quality(3VP)$.
>
> H_a: goal-based pattern description increases the quality of the produced model, $H_a : quality(GoF) < quality(3VP)$.

5.1.3. The Design task

For testing H_0 we planned two design tasks, each of these asks the subjects to produce a UML class diagram for modeling the architecture of a sub-system by employing a pair of design patterns. In delivering their UML diagrams,

the subjects are requested to fulfill a set of design goals and to obtain a set of quality concerns.

We envisaged two application contexts: the *BankLoanSystem* and the *RoboCup* problems. The first is a design problem in a bank context, in which the subject is asked to design the architecture for loan applications. The exercise prescribes the employment of *Template Method* and *Proxy* patterns. The RoboCup system is a design task in a context of robotic simulation, where the designer is asked to define a flexible architecture for coordinating a team of virtual robots; it recommends the use of *Mediator* and *Strategy patterns*.

5.1.4. Experiment design principles

In order to test H_0 we need to measure the quality of the UML class model produced by employment of patterns. In order to reduce the subjectivity in assessing the correctness, we prepared in advance a checklist for verifying whether a set of desired design goals or quality assets is satisfied in the delivered UML class diagram (the checklist was kept secret to the subjects). We balanced the two tasks thus to have 5 significant properties to be checked for each of them. Therefore checklists contain five entries for each task. Examples of entries of the checklist are: 1) "are the *checkCredit*, *checkStock* and *checkIncome* methods defined as abstract in the *BankLoanSystem* class and overridden in any of its subclasses (from the Template Method)?" 2) "does the *Coach* class implement a dynamic set of *Players* with register/unregister methods (from the Mediator)?".

Therefore the experimental design uses two independent variables that are the *exercise* and the *pattern documentation format* and one dependent variables that is the *correctness* of the solutions. This is measured by evaluating whether each entry in the checklist is fulfilled in the model (score=1) or not (score=0). The total score for this variable stays in the range $[0 : 5]$.

Other secondary-factors may influence the outcome of the experiment and we controlled them as follows. The lab: the order of the labs may have an effect (learning effect), as subjects who already received the treatment in the first lab may pay higher attention on requirements in the second lab. The system: since we use two systems, subjects could show different performances on different systems. So the system is also a secondary-factor.

We adopted a balanced design with two subsequent experimental sessions (called Lab 1 and Lab 2), each one during up to one hour in the same day. Subjects are randomly assigned to a group from A to D. The design ensures that subjects work on the two applications with the two treatments in all the possible permutations, as shown in Table 2. A pre-questionnaire have been administrated to profile the subjects, while a post-questionnaire allowed us to collect reactions to the exercise proposed in the Labs.

Co-factors, we cannot control but just measure, may also have influenced the experiment. Among them, in the profiling questionnaire, we measure the following aspects. Subjects' background: we identify two courses that are mandatory for the experiment tasks on patterns (e.g., Software Engineering and Java Programming). OO experience: when a subject already worked as programmer in big/real software projects with the use of object-oriented programming language, her/his experience is considered high, low otherwise; UML experience: when a subject already worked as designer in big/real software projects with the use of UML language, her/his experience is considered

Table 2: Design of the experiment.

Group	**Lab 1** (exercise/format)	**Lab 2** (exercise/format)
A	LoanSystem/GoF	RoboCup/3VP
B	RoboCup/3VP	LoanSystem/GoF
C	LoanSystem/3VP	RoboCup/GoF
D	RoboCup/GoF	LoanSystem/3VP

high, low otherwise. Design Pattern experience: when a subject already worked as programmer/designer in big/real software projects with the use of Design Patterns, her/his experience is considered high, low otherwise. In our case, as emerged from the responses to a pre-questionnaire administered to the subjects, very few of them had a limited experience in using design patterns.

We did not include the use of any tool in the experiment. The motivation is that selecting one of the (commercial) tools available for supporting GoF-style patterns into the design implies, for balancing the experiment, to extend it with a plug-in for supporting also the 3VP patterns. This would raise additional threats, difficult to control: (i) usability - are the two treatments equally supported by the CASE tool? (ii) Experience and learning effect - are the subjects already familiar with the instrument? is there a learning effect after the first lab? (iii) Internal Validity - is there an interaction between treatment and instrumentation? what do we measure, the documentation impact or the tool impact?

5.1.5. Analysis Method

In order to test the hypothesis H_0 we use a non-parametric test. Since we collect two measurements for each subject, data are intrinsically paired so we use a paired statistical test, the Wilcoxon one-tailed test. Such a test allows checking whether differences exhibited by the same subjects with different treatments over the two labs are significant.

The analysis of secondary-factors and co-factors is performed using a two-way Analysis of Variance (Anova) and, in case of interaction, it is visualized using an interaction plot. Although Anova is a parametric test, it is considered quite robust also for non-normal and non-interval scale variables [39].

5.1.6. Experimental Material

As already discussed, the experimental material consisted of two design tasks, one in the context of the *BankLoan* System and the second for the *RoboCup* System.

Figure 10 shows how the Task Description and the Checklist have been prepared. We have identified a design problem and selected two design patterns that could be used for solving that problem. Therefore, an analysis of possible forces to balance allowed us to define a list of properties the system must have. This list has been used for deriving (for each task) a description of the task (for the subjects) and a checklist for evaluating whether the desired property is fulfilled.

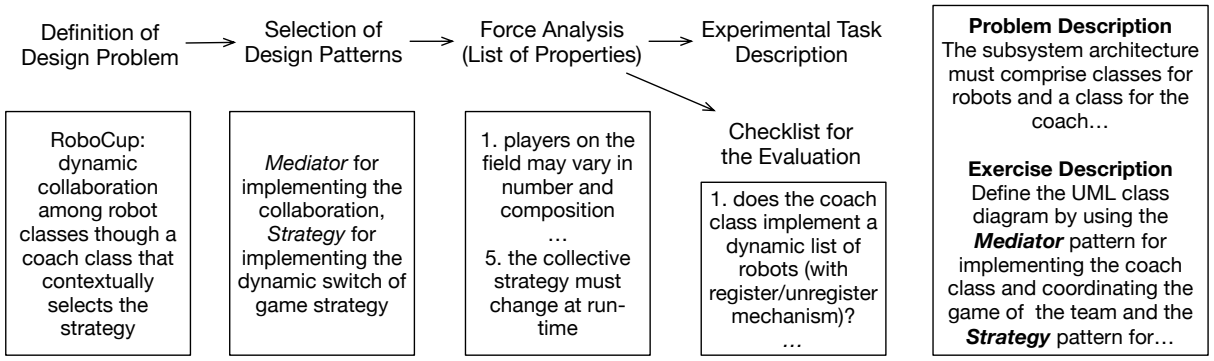


Figure 10: Description of the activities for contextually preparing the Task Description for the subjects and the Checklist for evaluating the correctness.

Only the description of the two tasks is provided to the subjects, and it is organized in two sections: i) the problem description illustrates the sub-system to be modeled, including the desired functions and class interactions; ii) the exercise description illustrates the suggested patterns and the non-functional qualities that are expected as an outcome (for instance "use the Mediator pattern for implementing a flexible collaboration mechanism among robot classes").

We also prepared the description of four patterns (Mediator, Strategy, Template Method and Proxy) in both the two documentation formats Gamma-like (GoF) and Goal-based (3VP).

Finally, in order to focus the design of the experiment we decided to let the subjects work only by handwriting using A4 papers and pencils. We also provided them with a common notation to use for unequivocally expressing their UML class diagrams. We have not provided any software design tool to the subjects to use in their task since it may represent a threat to validity (as already discussed in Section 5.1.4).

5.1.7. Operation and Execution

The experiment has been carried out in two parts, in two different days (see Table 3). The first day, subjects got trained with a lesson about the design patterns. The points focused in the training are: 1) how to read a GoF pattern, paying particular attention on applicability, consequences and implementing issues; 2) a simple exercise of employing a design pattern (specifically the Observer, documented in the GoF style); 3) how to read a goal-based pattern (the three views); 4) a simple exercise of employing the same design pattern but documented in the goal-based style.

Table 3: Activities and timing of the experiment execution.

	day 1	day 2				
activity	training	pre-quest	lab 1	break	lab 2	post-quest
timing	2 h	10 m	50 m	5 m	50 m	10 m

The second day we provided the subjects with an experiment pack, covering all the four phases, composed of the following material: a) pre-questionnaire, b) Lab 1 description, c) Lab 2 description, d) post-questionnaire. For

each lab we provided: i) a description of the design problem to solve, ii) an indication of the patterns to use, iii) the corresponding Gamma-like or Goal-Oriented description of patterns to be used for problem solution, iv) two blank sheets to draw the UML diagram to deliver.

The control of time has been kept by projecting a common clock in the room wall. We synchronized all the subjects by assigning 10 minutes for each questionnaire and 50 minutes for each lab. We also imposed a 5 minutes break between Lab1 and Lab2 to mitigate the fatigue effect. At the end of the scheduled times, all subjects returned us two questionnaires and two UML diagrams.

5.2. Results analysis

As a first step, descriptive statistics are used to visualize the collected data. Figure 11 shows the box plot of correctness for the two treatments (traditional vs goal-based pattern documentation). A clear trend is that the goal notation produces better results. This is confirmed by the paired analysis of correctness using the Wilcoxon one-tailed test (also known as Mann-Whitney's test). Table 4 shows a p-value very close to 0.035, therefore we can conclude that H_0 is rejected and H_a can be formulated. The negative difference shows that better results are obtained with the goal-based treatment, with a *medium* effect size (Cohen d effect).

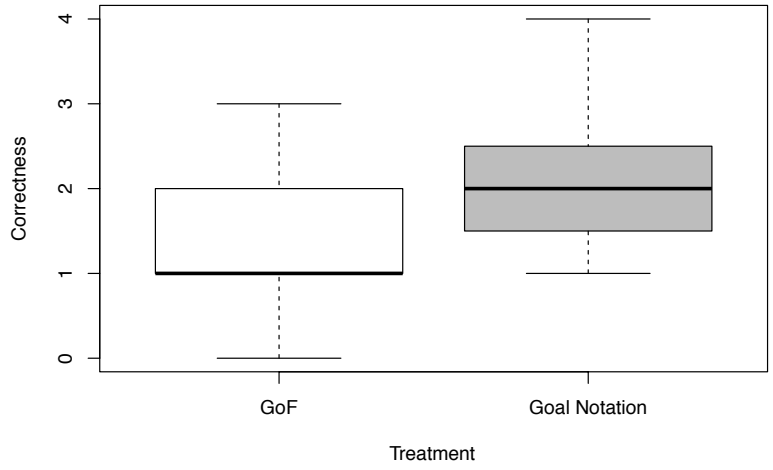


Figure 11: Box plot of correctness for the two treatments (traditional vs goal-based pattern documentation).

	name	mean	median	sd	p.value	effect.size
1	Gamma	1.25	1.00	0.87		
2	Goal	2.08	2.00	0.90		
3	Difference	-0.83	-1.00	1.11	0.0356	-0.7476

Table 4: Descriptive statistics and paired analysis (Wilcoxon one-tailed test).

Analysis of Secondary-factors and Co-factors. The analysis of secondary factors and co-factors is performed using the Analysis of Variance (Anova) and, in case of interaction, it is visualized using an interaction plot.

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
Treatment	1	4.17	4.17	4.90	0.0386
Lab	1	0.00	0.00	0.00	1.0000
Treatment:Lab	1	0.17	0.17	0.20	0.6627
Residuals	20	17.00	0.85		

Table 5: ANOVA of Correctness versus Lab.

Table 5 is the ANOVA table for the analysis of the Correctness versus the Lab and the Treatment. It shows that the secondary factor Lab did not significantly influence the dependent variable, thus, no learning effect could be observed between the two labs. In addition the interaction between Lab and the Treatment is not statistically significant; this also supports the thesis that the order of the treatments did not impact the observed results.

Also none of the co-factors has significantly influenced the correctness of the produced outcome. This is summarized in Table 6, 7, 8. In particular, having a high or low experience on object-oriented programming, or UML modeling did not influence the correctness of the produced outcome.

An interesting finding is the interaction between the experience in UML and the treatment (see Figure 12). Differently from expectation, the outcome of the exercise with GoF documentation is quite independent from the UML experience, whereas comparing the two curves, the greater is the knowledge on UML the better the results are with the new format of documentation.

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
Treatment	1	4.17	4.17	5.80	0.0258
OOExp	1	0.01	0.01	0.01	0.9296
Treatment:OOExp	1	2.78	2.78	3.87	0.0632
Residuals	20	14.38	0.72		

Table 6: ANOVA of Correctness versus OO Experience.

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
Treatment	1	4.17	4.17	5.90	0.0247
UMLExp	1	0.06	0.06	0.09	0.7623
Treatment:UMLExp	1	2.98	2.98	4.22	0.0532
Residuals	20	14.12	0.70		

Table 7: ANOVA of Correctness versus UML Experience.

5.3. Threats to validity

Here we discuss some of the main threats to the validity we envisage in the experiment. The classification of the threats is based on those described by Wohlin et al. [39].

Construct validity threats concern the relationships between theory and observation. They are mainly due to the method used to assess the outcomes of tasks. The proposed task requires a creative effort from subjects: they have

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
Treatment	1	4.17	4.17	5.07	0.0357
GoFExp	1	0.10	0.10	0.12	0.7275
Treatment:GoFExp	1	0.64	0.64	0.78	0.3874
Residuals	20	16.42	0.82		

Table 8: ANOVA of Correctness versus GoF Experience.

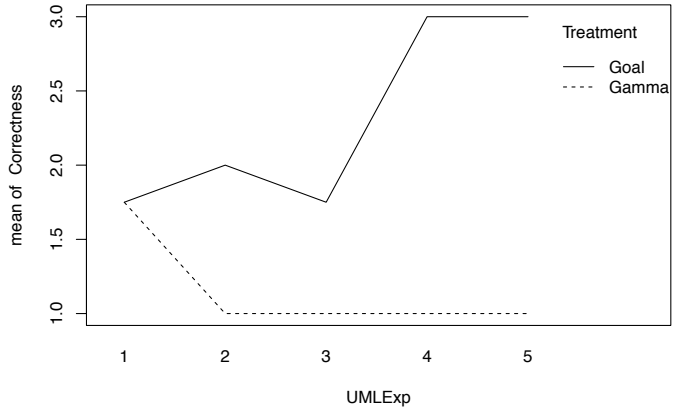


Figure 12: Interaction plot. Analysis of the interaction between Experience on UML and Treatment.

to build solutions to the design problem that can potentially differ from one another; anyway the structure of the resulting models should exhibit a set of properties that can be checked by the evaluators to rate the resulting quality. The proposed tasks have been prepared for including only GoF's patterns, therefore there is no evidence that results can be generalized to other categories of software patterns. The researchers who were present during the experiment ensured the duration of the various phases of the experiment.

Internal validity threats concern external factors that may affect a dependent variable. They may be due to the fatigue and learning effects between Labs, experienced by the subjects having them in sequence. The learning effect is mitigated by the experiment design we chose (a counter-balanced experiment design), while the fatigue effect is mitigated by the break we imposed between the two Labs. Another threat can be due to the fact that subjects were students and young researchers (mainly already working in our labs); so they were not selected randomly. Anyway, they were all very motivated to participate to the experiment because of their limited experience in the matter. At the same time this allowed us to have a group of individuals with quite homogeneous skills that have been further balanced through the training session on patterns documentation we organized the day before the actual experiment. We did not provide a CASE tool for supporting the design task, in order to avoid usability issues, learning effects and uncontrolled interaction between tool and treatment. Therefore there is no threat of instrumentation in this experiment. During the experiment, subjects were told that their grading in the course was not depending on the outcome of the task (design outcomes could be delivered in an anonymous form). It is the experiment facilitator's opinion that subjects felt a personal interest in the design exercise and the topic, so they made a serious attempt in their inspection.

Conclusion validity concerns the relationship between the treatment and the outcome. The statistical analysis is performed using a non-parametric test, Wilcoxon one-tailed test. It does not assume data normality and is also well suited for use on small samples. Moreover, two-way ANOVA was used mainly to detect possible interactions between each co-factor (e.g., application domain and labs schedule) and the Treatment.

External validity concerns the generalization of the results. The main threats, in this area, stem from the type of subjects. Subjects were students and young researchers with good experience in Software Engineering, close to finalize their education and to start a job in industry. They also have little experience in selecting and customizing patterns for a specific context. This allowed us to investigate the situation in which an individual has to be trained in, and also actually perform reuse practices, starting from a good knowledge on software design and Object Oriented programming. This is a quite frequent situation in the real practice, for example, when a young practitioner, having a culture that is comparable to that of a student, is employed in a software project [40]. So, the results of this study seem to be generalizable in this particular context.

Further studies and experiments should be performed to investigate whether or not our results can be generalized to more experienced subjects (e.g., professional software designer) having a deeper knowledge of the Gamma-like documentation and of patterns in general, when using the Goal-oriented format as a support to the reuse activity.

5.4. Interpretation and Discussion of Experiment Findings

Here we summarize our interpretations of the results of the experiment.

Subjects' degree of knowledge of the pattern does not impact the correctness of the produced result. The pre-questionnaire revealed that a few subjects already had a very limited experience in the use of patterns. However the analysis of co-factors revealed that background knowledge of the employed patterns had no impact into their employment. This means the new format is, at least, as effective as the traditional pattern format for providing pattern details. Although, given that the correctness is higher for the new format, it is possible to argue that the motivation for this result is that the pattern is understood in a shorter time with the new format.

Subjects' experience does not impact the correctness of the produced result. As long as subjects hold a certain level of experience in object-oriented programming and UML (in our case they were all students and young researchers in Computer Science) other differences in their background do not influence their ability to customize design patterns for a specific design context (see analysis of co-factors). We also verified something unexpected: those who have a greater experience in UML design produced better results with the use of the goal-based pattern documentation rather than the traditional one.

Explicitly providing decision points and trade-offs, thanks to the goal-oriented notation, helps the designer in producing a better customization of the design patterns for the specific problem context. The result of the experiment suggests that the new goal-based format encourages the pattern practitioner to focus on the reasoning process rather than on the solution, and this has an impact on the design outcome. By analyzing the outcome of the exercises, where the treatment was the GoF documentation, the produced UML diagram is in most of the cases, very similar to

the UML example shown in the book. We retain that the UML example reported as hook for designers represents a temptation for oversimplifying the solution. In our opinion, the designer is not encouraged in reading the whole pattern description (especially the section where all variants are described) and tries to reason only on the reported diagrams. This is not possible in the goal-based format, where there is no explicit UML diagram. The use of this format requires focusing on design goals and alternatives and to reason on the decisions to take and their consequences on the model.

Validating the process for reusing goal-oriented patterns. As mentioned before, the systematic reuse process is out of the scope of the empirical study, thus the second contribution of this paper remains un-validated. We are planning a new experiment in which subjects will be instrumented with two or more different processes. The crucial point to solve is that, so far, a standard process for pattern reuse does not exist. However, in literature there are many possible approaches, such as POAD [41], that could be used as alternative treatments for the experiment. In order to avoid internal threats to validity, subjects will be trained in the use of all the reuse processes to compare.

6. Final Remarks

This section reports some considerations concerning the expressiveness, the applicability and the authoring phase of the proposed goal-oriented method.

6.1. Expressiveness of the 3VPattern Documentation Format

In order to evaluate the ***scope of the 3VP notation*** we worked on the conversion of a selection of well-known patterns from the literature[3].

We currently limited the analysis to four categories of patterns: behavioral pattern, structural patterns and creational patterns (from the GoF's book [9]) and architectural patterns (from the POSA's book - volume I [1]). Three examples of these patterns are reported in the Appendix.

The first consideration concerns the possibility to shift category of patterns with a minimal impact on the notation and the overall approach. We experienced that shifting from classes-and-object towards components-and-interfaces was smooth and the semantics of the notation did not require particular changes. The explanation for this is that the notation does not directly include formalism for representing categories of elements that compose the solution. The Resource concept is a high level placeholder used to refer both to classes and objects and to components and interfaces without any specific change in the notation.

An extensive analysis of the use of the goal-oriented documentation to other categories of patterns (beyond the architectural and design patterns) is out of the scope of this paper and it is planned as future work. As a preliminary result, we report the use of the 3VPattern for documenting a pair of patterns from totally different domains.

[3] In order to share these patterns a wikiThe 3V Design Pattern Repository is available for the pattern community at `http://af.pa.icar.cnr.it/3vpattern/wiki`

The first example is a pattern for multi-agent systems [42] called *Embassy* which use is suggested to manage security mechanisms in a multi-agent organization and to facilitate the communications among different communities (Figure 13 shows the strategic-model). In this case the patterns deals with agents, communications and ontology.

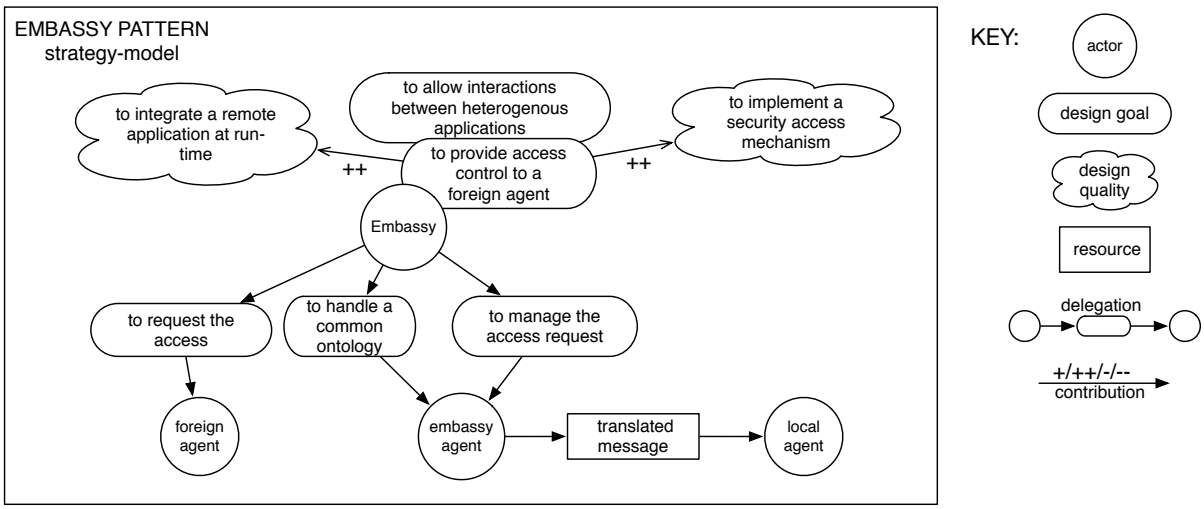


Figure 13: Example of strategic-model for the *Embassy* pattern from a repository of patterns for multi-agent system [42].

On the other hand Figure 14 shows the strategic-model of a workflow pattern [43]. This category of patterns deals with the organization of elements of a business process. In particular these patterns aim at capturing the various ways in which tasks and data are represented and utilized in workflows, in a form that does not rely on specific technologies. In particular, the *Block Data* is related to the extent and manner in which data elements can be viewed by various components of a workflow process. The elements of the solution are basically the process, the task and the data element. As shown in Figure 14 resources inside goals are provided as free text. This mechanism seems enough flexible to fit the change of paradigm without specific adaptation.

Anyhow, limiting to object-oriented design patterns and to architectural patterns the applicability of the approach is still significant. Indeed the notation can be employed to describe more than 350 different patterns [4].

Considerations about the expressiveness of the strategic model. The strategic diagram uses the abstraction of actor for referring to the participants and therefore to assign responsibilities. The representation has an informative content similar to that of CRC cards [44], which is generally suitable for representing collaboration among many entities. However a strategic diagram also indicates main system forces that are impacted (positively or negatively) by the pattern.

Compared to several approaches based on the UML notation, the proposed notation has some intrinsic advantages.

[4]This is the sum of the number of Architectural patterns and Object-Oriented Design patterns surveyed by Henninger and Correa in [6]. According the same source, Architectural patterns and Object-Oriented Design patterns constitute a majority of the types of development patterns (65%)

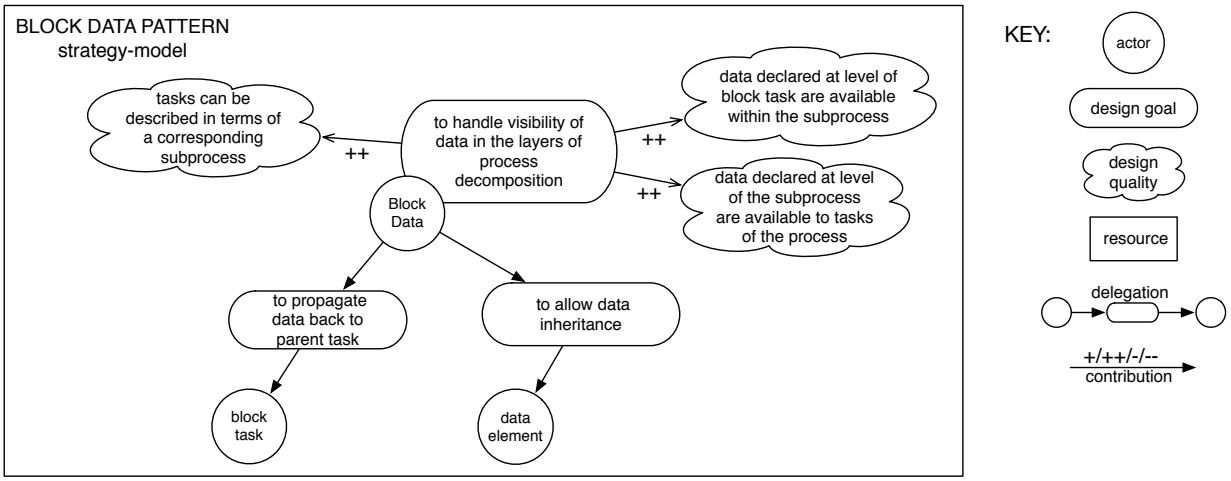


Figure 14: Example of strategic-model for the *Block Data* pattern from a repository of workflow patterns [43].

The strategic-model view provides an explicit structure that highlights intent and applicability of a design pattern. This view is a valid support for searching in a pattern catalogue, because the explicit semantics clarify the difference among some patterns that present similar structures. Let's take in consideration, for instance, the *State* and the *Strategy* patterns. This two patterns proposes a very similar static structure (see Figure 15), whereas the strategic-model view is sufficiently expressive for highlighting their totally different purposes.

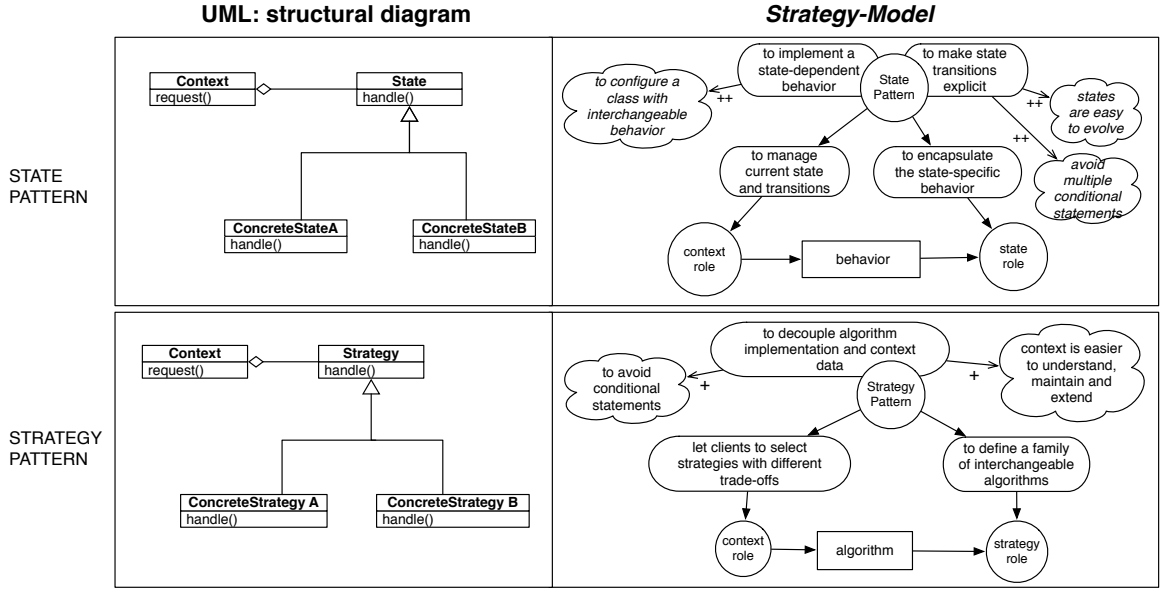


Figure 15: Semantic difference among *State* and *Strategy*.

Considerations about the expressiveness of the design-goal model. The design goal is an abstraction for representing recurrent design problems as first level citizen of the documentation. Goal models are conceived as a

systematic way for reasoning on a general problem by decomposing it in a simpler set of sub-problems. Exploring motivations and forces empathizes the activity of making contextual decisions.

This notation allows spanning a range of different categories of problems. For instance, when designing or developing an object oriented system typical problems are: finding appropriate objects, determining object granularity, specifying interfaces, identifying static relationships among classes, establishing dynamic interactions among objects.

Finally, the design-goal model provides a solution to the common problem of confusing a design pattern with a design template [15]. The UML notation is not appropriate to represent alternative solutions, because a diagram may show only one structure. Two mutually alternative solutions require two different diagrams. If a pattern contains many decision points, then the number of diagrams rapidly increases. Conversely, the design-goal model synthesizes many variants in a compact format thus to reduce this risk to confuse the example with the solution.

6.2. Some Notes on Authoring 3V-Patterns

This section provides some suggestions for converting a pattern from the classic format to a 3V-Pattern.

Guideline 1 - identify the actors: the name of the pattern is the main actor by default; then all the pattern participants are actors. Some design patterns have a further implicit actor (client) that represents classes that access to functionality of the roles: in these cases it is useful to make it explicit. For instance the Mediator pattern has three actors: the mediator pattern, the mediator role and the colleague role.

Guideline 2 - identify design goals: highlight all the sentences that define consequences of the pattern into the system, whether they are functionality or qualities of the system. When these sentences are identified then try to generalize them by using a shared vocabulary and the infinitive form. Examples of annotate design goals are shown in Figure 16: [distribution of behavior among objects] is revised into [to decentralize a complex behavior]. This activity suffers of the problem of lack of a common ontology. In order to reduce ambiguity and redundancy it is preferable to have many revisions, also considering other patterns.

Guideline 3 - put quality aspects in positive terms: this is a rule for supporting the coherence among quality assets of the system. The problem is that pattern descriptions sometime refer to a quality asset in positive terms (example [to provide simpler class]) whereas other times the same aspects is given in negative terms (example: [to reduce the complexity]). By default we prefer to use positive terms and, on the occurrence, to use a negative contribution to specify the negative form (for instance to [to increase the complexity] becomes a negative contribution towards [to provide simpler class]).

Guidelines 4 - look for variants and issues: in order to build the tree structure we use a mixed top-down bottom-up approach. Look at the two sections “Consequences” and “Implementing Issues” for all the possible variants of the pattern and their dependencies with quality aspects of the context. These variants are, very probably, the leaves of the tree. Then use a generalization to identify common points among these variants and decomposition in order to identify why these variants are connected to high level goals. For instance in the Mediator pattern description, the authors refer to two possible ways to manage interaction among colleagues and mediator (we called them [PUSH] and

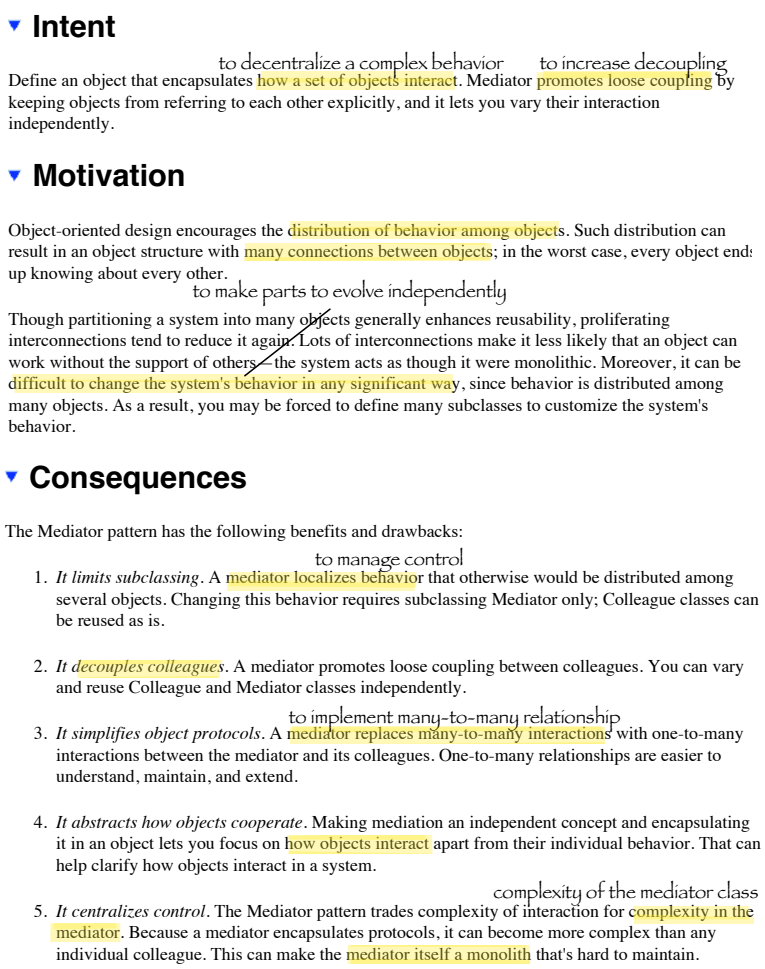


Figure 16: An example of annotation over the *Mediator* description extracted from the GoF's book [9].

[PULL] approaches). By reasoning on these two methods we generalized them as an answer to the problem of making the colleague classes indirectly communicate. So [to allow colleague to indirectly communicate] is connected to the root goal by an intermediate goal: [to identify how classes will interact].

7. Conclusions

This paper proposes a new format for documenting patterns that is based on strategic modeling, a conceptual framework originated in requirement engineering. The rationale of this choice is that when trying to understand and use a software pattern, it is often necessary to grasp the 'why' that underlie the 'how'. The new documentation format and the consequent reuse process is conceived for pushing the designer to focus on design decisions enclosed in design patterns rather than on the solutions. The empirical validation provided some interesting evidence that goes in the direction of confirming the hypothesis that the way the new documentation format conveys the information has a positive impact on the quality of the reuse process.

We also consider a series of open points that we discuss below.

Lack of a common ontology. The representation grounds on natural language to identify design goals, qualities of the system and design tasks. On one hand this maintains the original flexibility of the pattern description, but on the other hand it may be confusing. For instance the same design task may be proposed in different terms: [class A inherits class B] and [class is a subclass of class B] indicate the same relation between A and B. This problem is particularly relevant in the contribution analysis. For instance two patterns may have a (positive/negative) contribution to the same system quality, but they may use different terms for indicating it: [decrease coupling] and [increase class independence]. Being represented as two separated graphical elements, these may be considered different, whereas they are not. The problem would be solved by introducing a common ontology to use for disambiguate the natural language.

Understanding of the domain forces. The proposed approach aims at improving the understanding software designer needs to have of the properties of the system that a pattern will impact. It may happen that a designer is facing a complex problem and she has not yet gained a full understanding of all the components of the system and of all the forces that play a role in it. This is a relevant issue, because working with incomplete information from the domain may complicate the identification of the correct pattern to apply, and also the capability to customize it for the specific context. We have identified in a work from Gross and Yu [3] a way to overcome this issue. The solution, still in progress, is to complement them, having the two approaches put in sequence. The cited approach takes as input a set of requirements for the current system and it offers a systematic way to decompose them in a hierarchy of sub-goals and quality properties. Then, these design goals and the quality properties become an input for our approach: they are used to take design decisions and to customize the pattern for the specific context.

Testing other properties of the pattern format. Actually the experiment was conducted firstly for evaluating the correctness of the customization of a pattern for a specific problem context. Other experiments must be planned for testing other properties of the new format. In particular *learnability* may be tested with the following experiment. Subjects are provided with some patterns (in both formats) and they have a fixed time for studying them, than they have to answer to a set of questions about patterns' details. By measuring the time factor in the answering phase, it is possible to check what format is easy to learn.

Another interesting experiment could be conducted for testing the composition of goal-oriented patterns (blending activity). Even if the experiment in the previous section was already based on reusing two patterns in each laboratory, a more complete experiment should consider cases of conflicts among roles in the proposed patterns and the use of pattern composition techniques. Moreover, to assure the effectiveness of the experiment, the documentation formats must be carefully selected considering as possible treatment, for example, the POAD design models [19].

8. Acknowledgement

We would like to thank Mariano Ceccato for many hours of discussions on techniques for conducting and reporting experiments of software engineering.

References

[1] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, M. Stal, Pattern-Oriented Software Architecture, A System of Patterns, John Wiley & Sons Ltd, Chichester, England, 1996.

[2] U. Zdun, Systematic pattern selection using pattern language grammars and design space analysis, Software: Practice and Experience 37 (9) (2007) 983–1016.

[3] D. Gross, E. Yu, From non-functional requirements to design through patterns, Requirements Engineering 6 (1) (2001) 18–36.

[4] I. Araujo, M. Weiss, Linking Patterns and NonFunctional Requirements, in: Proceedings of the Ninth Conference on Pattern Language of Programs (PLoP 2002), September 8–12, 2002, 2002, pp. 8–12.

[5] S. Yacoub, H. H. Ammar, A. Mili, Constructional design patterns as reusable components, in: Software Reuse: Advances in Software Reusability, Springer, 2000, pp. 369–387.

[6] S. Henninger, V. Corrêa, Software pattern communities: Current practices and challenges, in: Proceedings of the 14th Conference on Pattern Languages of Programs, ACM, 2007, p. 14.

[7] L. Rising, The pattern almanac, Addison-Wesley Longman Publishing Co., Inc., 2000.

[8] M. Cline, The pros and cons of adopting and applying design patterns in the real world, Communications of the ACM 39 (10) (1996) 47–49.

[9] E. Gamma, R. Helm, R. Johnson, J. Vlissides, Design Patterns: Elements od Reusable Object-Oriented Software, Addison-Wesley Professional Computing Series, Addison-Wesley Publishing Company, New York, NY, 1995.

[10] W. M. van Der Aalst, A. H. Ter Hofstede, B. Kiepuszewski, A. P. Barros, Workflow patterns, Distributed and parallel databases 14 (1) (2003) 5–51.

[11] D. C. Schmidt, M. Stal, H. Rohnert, F. Buschmann, Pattern-Oriented Software Architecture, Patterns for Concurrent and Networked Objects, Vol. 2, John Wiley & Sons, 2013.

[12] M. Schumacher, E. Fernandez-Buglioni, D. Hybertson, F. Buschmann, P. Sommerlad, Security Patterns: Integrating security and systems engineering, John Wiley & Sons, 2013.

[13] UIUC, Pattern Stories Wiki, Avaiable at http://c2.com/cgi/wiki?PatternStoriesWiki.
URL `http://c2.com/cgi/wiki?PatternStoriesWiki`

[14] P. Iaría, U. Chesini, Refining the observer pattern: The middle observer pattern, in: Proceedings of PLoP'98 (Group 7 Scattered Work), 1998, pp. 1–6.

[15] D. Riehle, Lessons learned from using design patterns in industry projects, in: Transactions on pattern languages of programming II, Springer, 2011, pp. 1–15.

[16] L. Prechelt, B. Unger-Lamprecht, M. Philippsen, W. Tichy, Two controlled experiments assessing the usefulness of design pattern documentation in program maintenance, Software Engineering, IEEE Transactions on 28 (6) (2002) 595–606.

[17] M. Duell, Managing change with patterns, in: Design patterns in communications software, Cambridge University Press, 2001, pp. 251–258.

[18] T. Mikkonen, Formalizing design patterns, in: Proceedings of ICSE '98, IEEE Computer Society, Washington, DC, USA, 1998, pp. 115–124.

[19] S. Yacoub, H. Ammar, UML Support for Designing Software Systems as a Composition of Design Patterns, Uml 2001: The Unified Modeling Language: Modeling Languages, Concepts, and Tools: 4th International Conference, Toronto, Canada, October 1-5, 2001: Proceedings.

[20] J. McPhail, D. Deugo, Deciding on a pattern, Lecture Notes in Computer Science (2001) 901–910.

[21] N. Harrison, P. Avgeriou, U. Zdun, Using patterns to capture architectural decisions, Software, IEEE 24 (4) (2007) 38–45.

[22] E. S.-K. Yu, Modelling strategic relationships for process reengineering, Ph.D. thesis, University of Toronto, Toronto, Ont., Canada, Canada (1996).

[23] D. Riehle, Bureaucracy, Pattern Languages of Program Design 3 (1998) 163–186.

[24] E. Wallingford, Sponsor-selector, Addison-Wesley Software Pattern Series (1997) 67–78.

[25] A. Eden, Y. Hirshfeld, A. Yehudai, LePUS-a declarative pattern specification language, Techn. rep 326 (1998) 98.

[26] D. Mapelsden, J. Hosking, J. Grundy, Design pattern modelling and instantiation using dpml, in: Proceedings of CRPIT '02, Australian Computer Society, Inc., Darlinghurst, Australia, Australia, 2002, pp. 3–11.

[27] L. Sabatucci, M. Cossentino, A. Susi, Introducing Motivations in Design Pattern Representation, in: Formal Foundations of Reuse and Domain Engineering: 11th International Conference on Software Reuse, ICSR 2009, Falls Church, VA, USA, September 27-30, 2009. Proceedings, Springer, 2009, p. 201.

[28] J. Mak, C. S. T. Choy, D. Lun, Precise specification to compound patterns with ExLePUS, Proceedings of the 27th Annual International Computer Software and Applications Conference (COMPSAC 2003).

[29] D. Kim, R. France, S. Ghosh, E. Song, A UML-Based Metamodeling Language to Specify Design Patterns, Proc. Workshop Software Model Eng.(WiSME) with Unified Modeling Language Conf.

[30] R. B. France, D.-K. Kim, S. Ghosh, E. Song, A uml-based pattern specification technique, IEEE Trans. Softw. Eng. 30 (3) (2004) 193–206. doi:http://dx.doi.org/10.1109/TSE.2004.1271174.

[31] G. Sunyé, A. L. Guennec, J.-M. Jézéquel, Design patterns application in uml, in: ECOOP '00: Proceedings of the 14th European Conference on Object-Oriented Programming, Springer-Verlag, London, UK, 2000, pp. 44–62.

[32] J. Mak, C. Choy, D. Lun, Precise modeling of design patterns in UML, in: Proceedings of the 26th International Conference on Software Engineering, IEEE Computer Society Washington, DC, USA, 2004, pp. 252–261.

[33] J. Dong, Uml extensions for design pattern compositions, Journal of Object Technology.

[34] J. Vlissides, Tooled composite, C++ Report (1999) 43–47.

[35] G. Larsen, Designing component-based frameworks using patterns in the uml, Commun. ACM 42 (10) (1999) 38–45. doi:http://doi.acm.org/10.1145/317665.317674.

[36] B. Chandrasekaran, Design problem solving: A task analysis, AI magazine 11 (4) (1990) 59.

[37] P. Bresciani, P. Giorgini, F. Giunchiglia, J. Mylopoulos, A. Perini, Tropos: An agent-oriented software development methodology, Autonomous Agents and Multi-Agent Systems 8 (3) (2004) 203–236.
URL `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.1.7049`

[38] D. Riehle, Describing and composing patterns using role diagrams, in: K.-U. Mätzel, H.-P. Frei (Eds.), 1996 Ubilab Conference, Zürich, Germany, 1996, pp. 137–152.
URL `citeseer.ist.psu.edu/riehle96describing.html`

[39] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, A. Wesslén, Experimentation in software engineering, Springer, 2012.

[40] M. Höst, B. Regnell, C. Wohlin, Using students as subjects—a comparative study of students and professionals in lead-time impact assessment, Empirical Software Engineering 5 (3) (2000) 201–214.

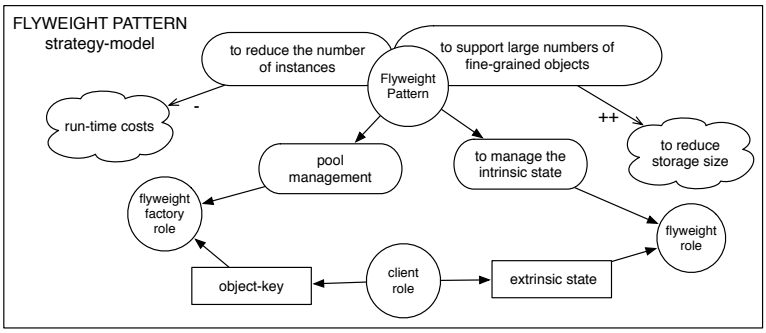
[41] S. M. Yacoub, H. H. Ammar, Pattern-oriented analysis and design: composing patterns to design software systems, Addison-Wesley Professional, 2004.

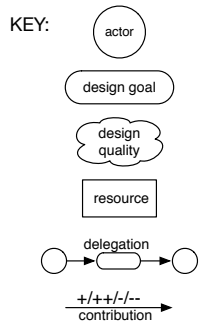
[42] S. C. Hayden, C. Carrick, Q. Yang, A catalog of agent coordination patterns, in: Proceedings of the third annual conference on Autonomous Agents, ACM, 1999, pp. 412–413.

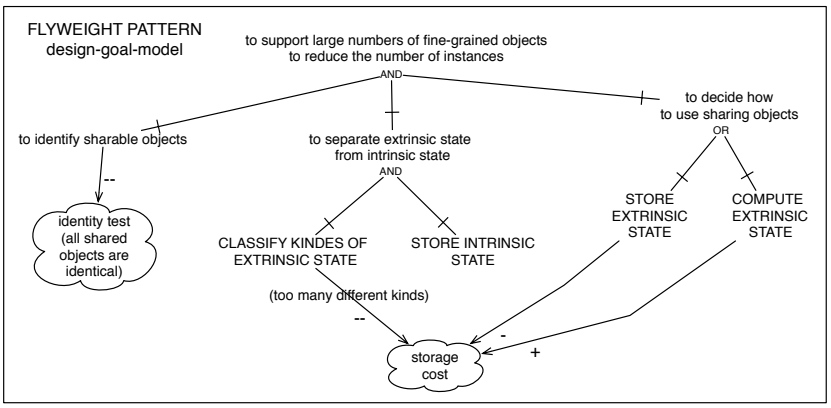
[43] N. Russell, A. H. Ter Hofstede, D. Edmond, W. M. van der Aalst, Workflow data patterns, Tech. rep., QUT Technical report, FIT-TR-2004-01, Queensland University of Technology, Brisbane (2004).

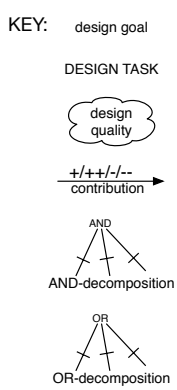
[44] K. Beck, W. Cunningham, A laboratory for teaching object oriented thinking, in: ACM Sigplan Notices, Vol. 24, ACM, 1989, pp. 1–6.

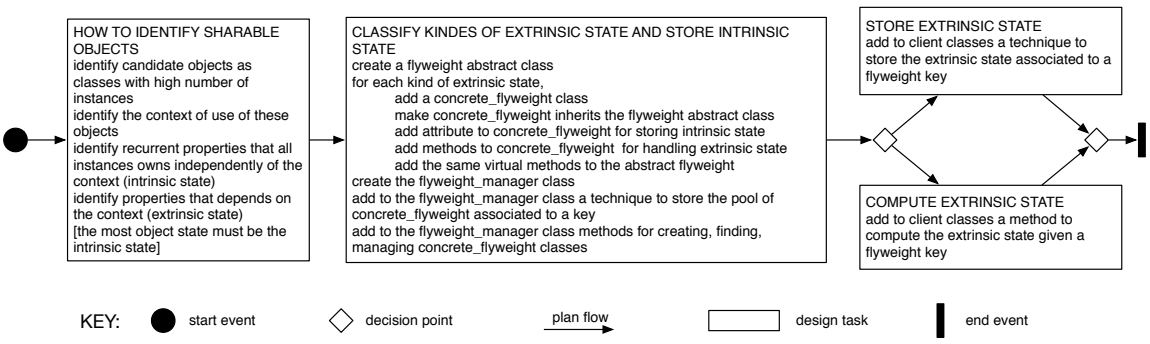
Appendix A. Flyweight Pattern



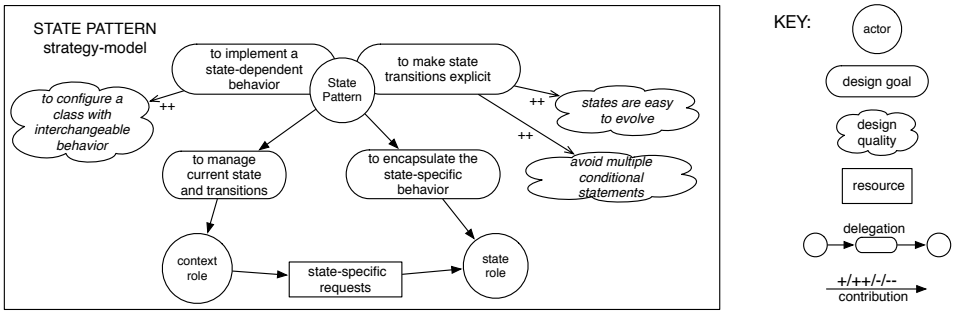


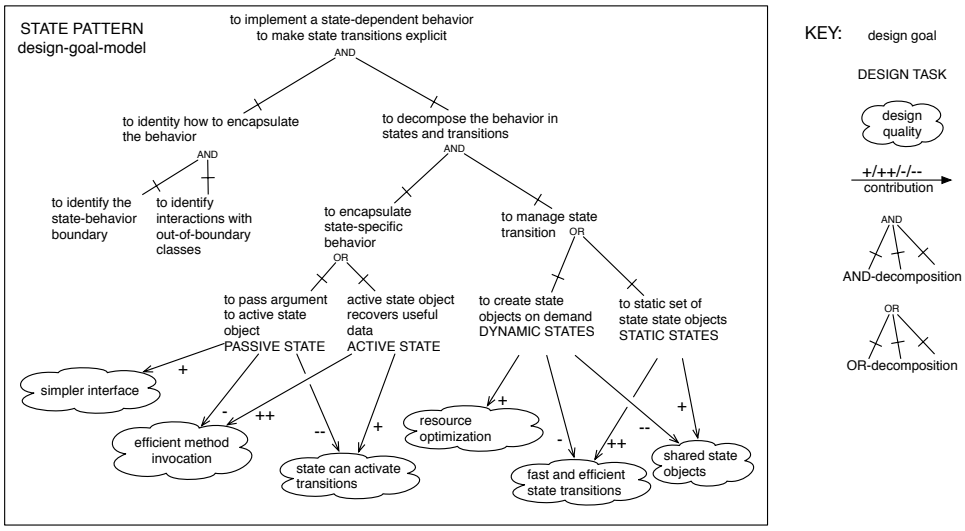


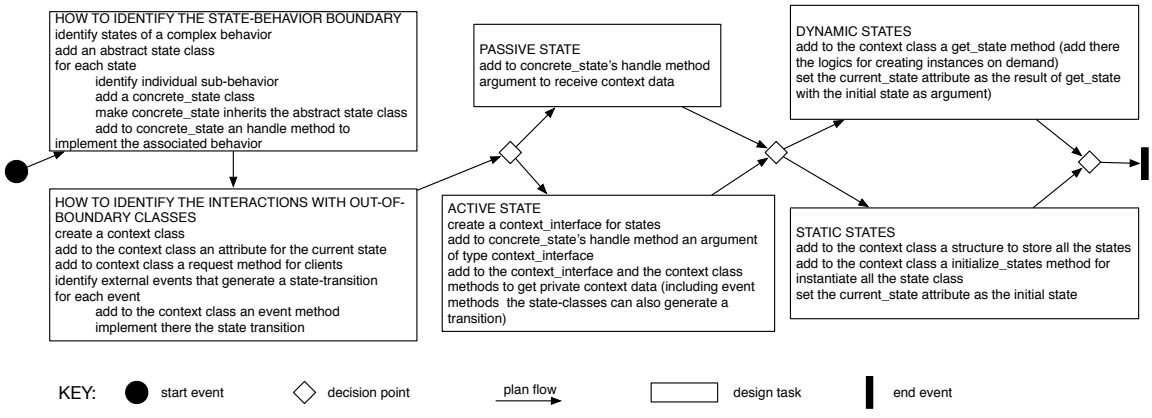




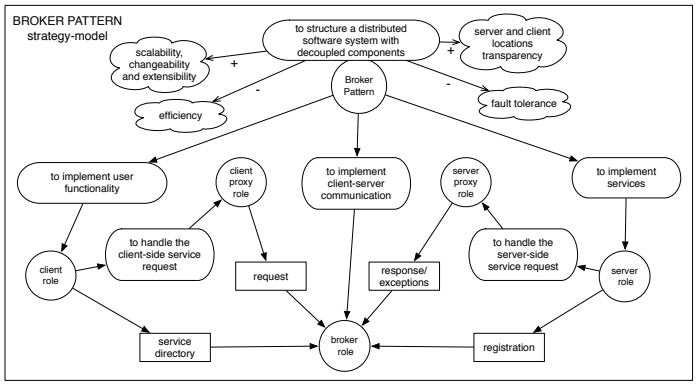
Appendix B. State Pattern

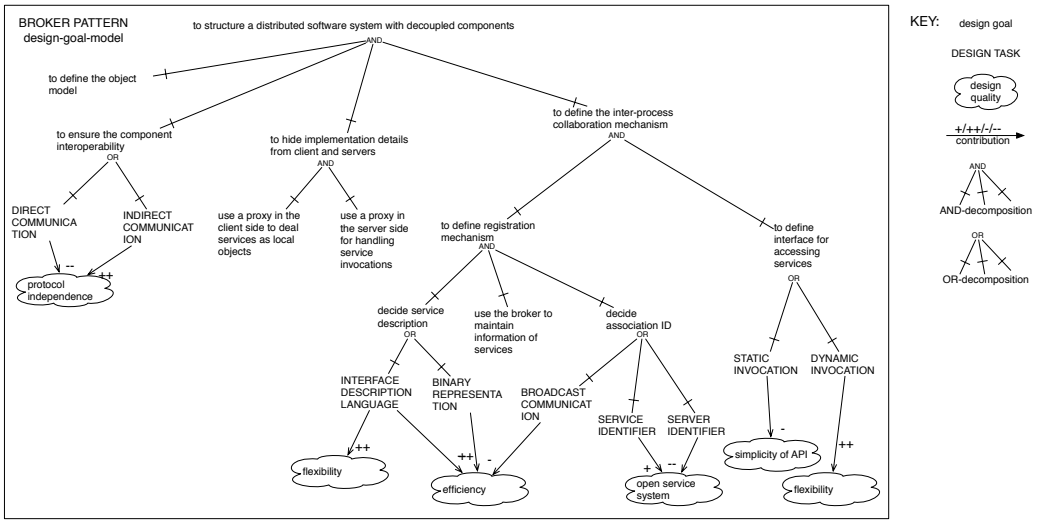


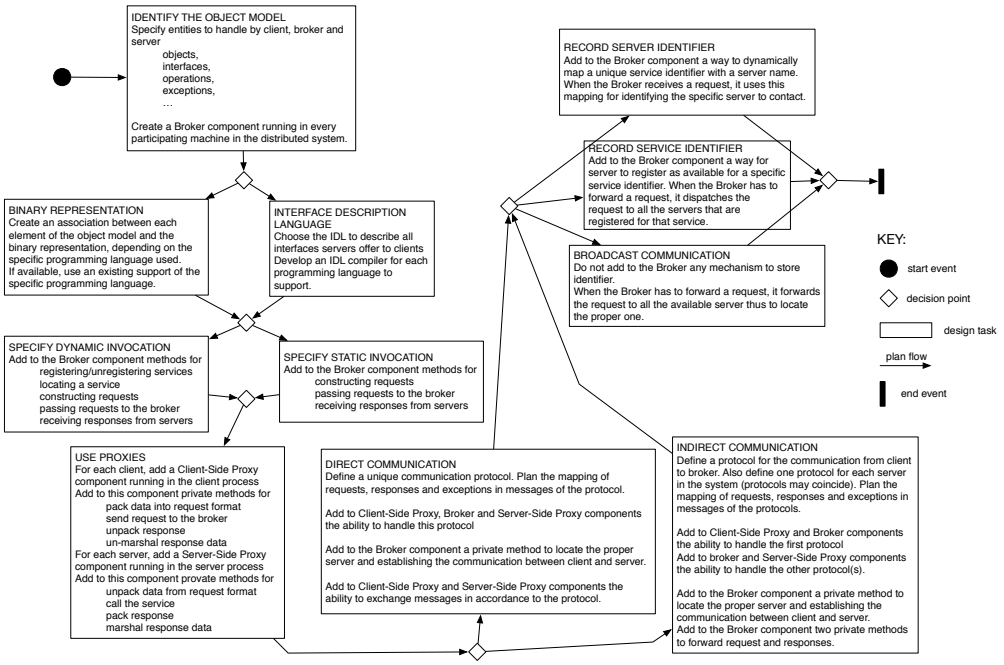




Appendix C. Broker Pattern







Appendix D. Checklists for Evaluating the Correctness

This section reports an extract of the design of the experiment presented in this paper.

Exercise 1: banking loan application. The exercise requires to define the UML class diagram for implementing the suggested architecture (a sequence diagram is provided) for bank loan applications in which the *Template Method* pattern to define a flexible way for encapsulating the general procedure for loan application, and allow a set of alternating concrete procedures by sub-classing and the *Proxy* pattern for abstracting the communication with a remote bank.

Evaluators must provide a score (from 0 to 1) to each of these item in the checklist (considering that names of classes or methods may slightly change in each delivered class diagram):

1. are checkBank, checkLoan and takeLoan methods factored in the BankLoanSystem class?
2. are checkCredit, checkStock and checkIncome methods defined as abstract in the BankLoanSystem class and then overridden in any subclass?
3. is there only one generic proxy class and many RealBank classes?
4. is the PrivateLoanSystem class able to access to an array of proxy objects, each of these referring to a RealBank objects?
5. is the checkBankCredit method abstract in the Bank class and then overridden in BankProxy class and in all RealBank sub-classes?

Exercise 2: Robocup Soccer Team. The exercise requires to define the UML class diagram for implementing the suggested architecture (three sequence diagrams are provided) in which each robot and the coach are modeled as classes. The *Mediator* pattern is suggested for implementing the collaboration among the robots of the team through their coach. The mechanism must be flexible enough to allow teams with different number of players. The *Strategy* pattern allows the coach to change game strategy among at least three different game strategies to implement in separated classes.

Evaluators must provide a score (from 0 to 1) to each of these item in the checklist (considering that names of classes or methods may slightly change in each delivered class diagram):

1. is there a bidirectional reference between players and coach?
2. are the methods stay, move, follow, pass, kick public operations of the Player class, whereas methods ball position, opponent position are public operations in the Coach class?
3. does the Coach class implement a dynamic set of players with register/unregister methods?
4. does the Coach class get a current_strategy attribute (or analogue) typed as GameStrategy?
5. does the GameStrategy class and all its sub-classes get a private method to communicate the strategy to each Player class that is on the field?

*Biography

Luca Sabatucci is a research scientist in the Agent-Oriented Software Engineering unit of the Italian National Research Council of Italy (CNR) since 2011. His research interests are in the areas of Self-Adaptive Systems, Requirements Engineering, and Design Patterns. He is the author of about fifty papers published in scientific journals, conferences and workshops. He participated in the organization committee of RE '11 and in program committees of many international conferences and workshops.

Massimo Cossentino got his PhD in Computer Science Engineering from the University of Palermo and his Habilitation à Diriger des Recherches (HDR) from the University Paul Sabatier of Toulouse in 2008. He is a Research Scientist of the National Research Council of Italy from 2001. In 2006-2008 he has been an invited Associate Professor at the University of Belfort-Montbelliard (UTBM). He is currently researching on Agent-Oriented Software Engineering, more specifically on adaptive workflows, simulation of traffic and transportation systems, design methodologies. He is the author of about one hundred and forty papers published in scientific journals, conferences and workshops. He is currently chairing the IEEE FIPA Design Process Documentation and Fragmentation Working Group. In the past he organized and chaired several international scientific events. He also got funded some national projects where he was the principal investigator and he participated in several others.

Angelo Susi is a research scientist in the Software Engineering unit of FBK. His research interests are in the areas of Requirements Engineering, Goal-oriented software engineering, Formal Methods for requirements validation, and Search-based software engineering. He published more than 80 refereed papers in journals and international conferences and participated in the organization committee of several conferences, such as SSBSE '12 (General Chair), RE '11 (Local and Financial chair) and in program committees of international conferences and workshops (such as AAMAS, ICSOC, CAiSE and SSBSE). He participated to several funded projects and he is currently acting as Scientific Manager of the FP7 RISCOSS project.