Introducing Motivations in Design Pattern Representation

Luca Sabatucci[1], Massimo Cossentino[2], and Angelo Susi[1]

[1] Fondazione Bruno Kessler IRST, Via Sommarive, 18 I-38050 Trento, Italy
sabatucci,susi@fbk.eu
[2] ICAR-CNR, Consiglio Nazionale delle Ricerche, Palermo, Italy
cossentino@pa.icar.cnr.it

Abstract. Design pattern formalization is aimed at encouraging the use of design patterns during the design phase. Many approaches focuses on providing solutions with a graphical notation and complementary text, typically composed by a static and a dynamic definitions. The weak point is the lack of flexibility when customizing the generic solution to the specific context of use. This paper proposes a criterion to motivate design pattern selection and reuse. Designer is supported with a technique for balancing pattern and context forces for selecting among alternative implementations. The provided representation summarizes and organizes relevant information in the classical informal pattern documentation.

Key words: Design Patterns, Goal-Oriented Modeling

1 Introduction

The informal description provided by the Gamma et al. (GoF) book [**?**] is very rich of details and it is perfectly suitable for communicating successful experience. Despite of the clarity of the exposition and the leading example that gives further clarifications and improves the global understanding, this format is not the best way to quickly and properly handle a pattern at design time. The description is long-winded and many important information are scattered along subsections (about 10 pages for each pattern) thus the reasoning process on the design problem is not properly supported. In addition, during the design phase many implementing details can be lost thus patterns gets poorer than in the original intent.

In order to better handle pattern complexity, many representations in literature use the concept of pattern role [**?**,**?**,**?**]. Originally conceived as a shortcut to talk about generic collaborating elements [**?**], the pattern role turned into an holder of responsibilities [**?**], thus drawing up patterns to social organizations [**?**]. This concept is the base for some preliminary works on design patterns, such as [**?**] where a catalogue of design patterns for the agent oriented design was presented and [**?**] dealing with design pattern composition. These previous experiences raised the need for an instrument for creating a representation where original design patterns topics, such as motivations, applicability, consequences and implementing issues, are maintained but where information is in a compact and handleable form.

The aim of this work is to propose a semi-formal representation for design patterns that considers motivations underlying the solution structure as the basic key for describing and reusing design patterns. This representation is different from many approaches in literature, that provide the detailed specification of "what" is to be done when reusing a pattern. Here the pattern is primarily concerned with exposing "why" certain choices for behaviour and/or structure were made or constraints introduced. The approach is based on the i* framework, that uses goal-oriented analysis for modeling and reasoning on strategic relationships among multiple actors of a domain. The approach derives from a mapping between the design pattern domain and the goal-oriented analysis, thus creating a framework for reasoning on force balancing, thus allowing the designer to better understand and customize the solution for the specific reuse context.
The proposed representation summarizes and reorganizes information taken from the informal textual description with the benefit to present relevant data in a compact format. The semi-formal nature of the representation is due to two factors: (i) the i* framework is provides a not fully semantically formalized language for defining system requirements, and (ii) a close connection is maintained between the i* pattern formalization and its informal textual description.

The paper is organized as follows: Section **??** provides a brief introduction of the i* framework. Section **??** describes the representation, from designer's need to the implementing solution. Section **??** discusses the approach, reasoning on benefits, understandability and reuse issues. Some related work are analyzed and compared in Section **??**, and finally some conclusions are given in Section **??**.

2 Background: The i* Framework

The i* framework [**?**] supports goal-oriented modeling and reasoning about functional and non-functional requirements. It is a conceptual framework for modeling social domains (in which both humans and software systems coexist) and provides constructs for expressing concepts that appear during the requirement process: actors, intentional elements (such as, goals, softgoals, tasks and resources) and relationships among those concepts. A relevant characteristic of the approach is that of offering means for representing not only the requirements of the system to be, but also the motivations for underling design choices. The intentional elements and the relationships between them allow answering questions such as why particular behaviours, informational and structural aspects have been chosen to be included in the system requirements, what alternatives have been considered, what criteria have been used to deliberate among alternative options, and what are the reasons for choosing one alternative against the other. This representation supports the analysis of strategies, which help reach the most appropriate trade-offs among (often conflicting) goals and soft-goals. A strategy consists of a set of intentional elements that are given initial satisfaction values.

Actors are holders of intentions; they are the active entities in the system or its environment who want goals to be achieved, tasks to be performed, resources to be available and softgoals to be satisfied. Actors in a system collaborate in order to address common goals.

Intentional Elements. The *Goal* is a condition or state in the world that actors would like to achieve. How the goal is to be achieved is not specified, allowing alternatives to be considered. The *Soft Goal* is similar to a goal, but there are no clear-cut criteria for whether the condition is achieved, and it is up to subjective judgment and interpretation of the modeler. Soft goals are often used to describe qualities and non-functional aspects such as security, robustness, performance and usability. The *Task* specifies a particular way of doing something. Tasks can also be seen as the solutions in the target system, which will address (or operationalize) goals and softgoals. These solutions provide operations, processes, data representations, structuring, constraints, and agents in the target system to meet the needs stated in the goals and soft goals. The *Resource* represents a physical or informational entity, for which the main concern is whether it is available. Actors and intentional elements are connected by different types of structural and intentional relationships. Several modelling perspectives allow to specify relationships between concepts.

Actor Modeling Perspective focuses on the identification of actors that participate to a social organization. This activity mainly concerns on identifying high-level dependencies relationships among actors. 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 expressed by an intentional element thus to specify the nature of the dependency and its motivation.

The ***Goal Modeling Perspective*** is focused on detailing an actor's boundary, defining its intentional elements according to various techniques. The *Decomposition analysis* allow to refine the goals or plans into sub-goals or subplans generating a goal/plan hierarchical decomposition. In the AND decomposition, all of the decomposing intentional elements are necessary for the target intentional element to be satisfied. Whereas OR decomposition provides a description of alternative ways of satisfying a target intentional element. The *Means-ends analysis* allows to represent the operationalization of a goal via a task via the specification of *means-ends* relationships. The *Contribution analysis* defines the level of impact that the satisfaction of a source intentional element has on the satisfaction of a destination intentional element. The i* framework defines a standard set of contributions: "$--$"/"$-$", strong/weak negative, the intentional element is sufficiently/partially dissatisfied; "$++$"/"$+$", strong/weak positive, the intentional element is sufficiently/partially satisfied.

3 Pattern Intentions: from Purpose to Solution

The introduction of the GoF's book [**?**] reports "*Design patterns goal is to capture design experience in a form that people can use effectively. Design patterns help you choose design alternatives that make a system reusable and avoid alternatives that compromise reusability*". The context gives a dimension of pattern usability, by providing motivations that justify its reuse and consequences of its application into the system. This is classically done by using a leading example that illustrates a concrete design problem and how the class and object structures in the pattern can solve the problem.

Goal-oriented analysis can be profitably employed to describe pattern motivations, forces and consequences in a semi-formal way, as well as Gross and Yu already did

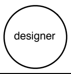
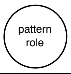
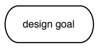
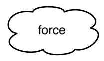
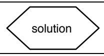
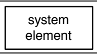
Domain element	i* element	Glyph	Rationale
Designer	Actor		He/she is the main actor, responsible of modeling the system, balancing forces and handling trade-offs.
Pattern role	Actor		It is an intentional element, holder of the responsibility to manage a piece of the pattern solution.
Design goal	Goal		It represents a design intention that is gathered by the designer and demanded to the pattern.
Force	Soft-Goal		Non-functional requirement that gives a guideline to choose among several solution implementations.
Solution	Task		It represents a step of the plan to apply the solution to the system.
System element	Resource		It represent one of elements for which the designer is concerning.

Table 1. Association map between pattern domain terms and i* concepts used in this work.

in [**?**] even if limited to non-functional requirements. Here, the proposed approach is based on abstracting the process of modeling a system with the support for reusing design patterns.

3.1 Designer Needs

The proposed abstraction considers the designer as the main Actor of the design activity domain, whose job is to balance design forces coming from the system under modeling. The pattern reuse activity elicits from designer's needs that are: (i) *Design Goals* to solve specific design problem emerging during the development of the system related to the correct distribution of responsibilities among classes of the system and (ii) *Non-Functional Requirements* (soft goals) that emerge during the analysis phase and specify qualities of the system-to-be. These needs define some conditions in the model that designer would like to achieve; non functional requirements are similar to designer's needs, but there are no clear-cut criteria for whether conditions are achieved.
The concept of *Pattern Role* is the core of the proposed representation. Riehle (in [**?**]) introduces role diagrams that focus on the collaboration and distribution of responsibilities between objects of the system. Roles are holder of responsibilities whereas the notion of class becomes an implementation construct only. The current work proposes to enrich this concept of role by giving it the responsibility to handle a piece of the pattern solution. This upgrades the role from a passive template for the solution, to an active reasoning element for achieving the solution. In this vision, a design pattern is the delegation of some design choices to the experience of expert designers. It is composed by:

(i) *Actors*, that represent active entities of the system who want goals to be achieved, tasks to be performed, resources to be available and softgoals to be satisfied. The abstraction considers the designer as the main actor of this domain, which job is to balance

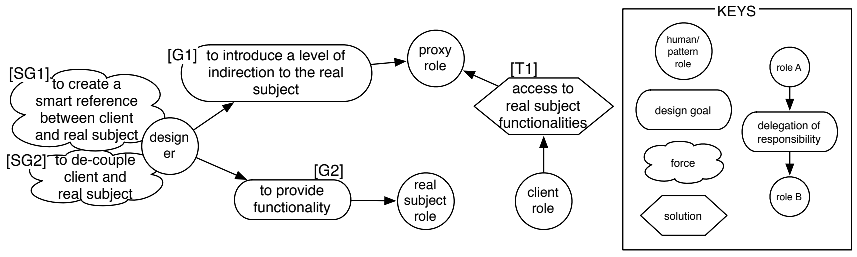

[SG1] to create a smart reference between client and real subject
[SG2] to de-couple client and real subject
designer
[G1] to introduce a level of indirection to the real subject
proxy role
[G2] to provide functionality
real subject role
[T1] access to real subject functionalities
client role
KEYS
human/pattern role
design goal
force
solution
role A
delegation of responsibility
role B

Fig. 1. Actor diagram for the Proxy pattern

forces coming from the context in order to address some design (functional/non functional) objectives. The pattern contains some roles, that represent the proactive parts that are used to allocate the solution to elements of the context.

(ii) *Goals* encapsulates the intentional part to solve the problem. A design goal is a condition or state in the model that designers would like to achieve; how the goal is to be achieved is not specified, allowing alternatives to be considered.

(iii) *Soft Goals* are similar to goals, but there are no clear-cut criteria for whether the condition is achieved, and it is up to subjective judgment and interpretation of the modeler. Softgoals are used to describe forces coming from the context that represent specific qualities of the system-to-be.

(iv) *Tasks* specify a particular way of doing something. Tasks are atomic component of pattern solutions, addressing design goals and soft goals. Tasks provide operations, processes, data representations, structuring, constraints to meet the needs stated in the goals and softgoals. The whole pattern solution is the synthesis of actions, guidelines and techniques described in pattern tasks.

(v) *Resources* represent physical or informational entities of the system, for which design activity is executes. Resources in a pattern are elements of the system such as data to introduce into the system in order to obtain a good solution.

Summarizing, the proposed map considers a pattern role as an actor that encapsulates a piece of the well-tested experience of an expert. The i* framework models this situation so that, when reusing a pattern, the designer delegates a part of his/her duty to pattern roles. Thus each role addresses some goals and proposes a strategic plan to achieve them. The actor diagram is the instrument to represent the pattern collaboration view. Table **??** gives an outline the mapping between terms of these two domains.

Figure **??** depicts responsibility organizations for the Proxy pattern. The i* visual notation represents actor as circles having a balloon associated to represent its internal rationale; design goals are represented by using rounded rectangles, whereas clouds are used to specify context forces. The main actor is always the designer of the system, who orchestrates with pattern roles, by delegating them some design responsibilities. Delegation is a dependency represented as an intentional element (goal, soft-goal, task or resource) that connect two actors. The directions of arrows indicate who is the original handler and who is the receiver of the responsibility. As an instance, in the Proxy pattern, the designer needs *[to create a smart reference between two objects]* (SG1) and *[to de-couple these two objects]* (SG2); this couple of soft-goals represents the main motivation for the use of this pattern. The commitment of these objectives requires a delegation of responsibility: the proxy is responsible *[to introduce a level of indirection between these two objects]* (G1) and the real subject must be able *[to provide the functionality]* (G2). Finally a third role, the client delegates *[the access to real subject functionality]* (T1) to the proxy.

3.2 Alternative Solution Implementations

Design patterns are descriptions of communicating objects and classes that are customized to solve a general design problem in a particular context. One of the common limits of most pattern representation techniques in literature is that solutions are provided as rigid templates that solve problems in a unique and invariant way. A pattern description should maintain its original informative content: applicability, pitfalls, hints, criticism and, above all, implementing alternatives and design issues.

This informative core is maintained by using the i* framework goal modeling perspective. This perspective uses goal decomposition to explore motivations for each specific implementing detail. Each role as at least a main goal to address and a collection of forces concerning qualities of the system. The decomposition analysis is performed by using AND/OR and means-end analysis techniques that generate a hierarchy of goals, sub-goals, tasks and resources for addressing role main goals.

Goal Model Hierarchy. Role main goals describe motivations for applying a specific pattern into a context. By using the AND decomposition, goals are iteratively refined in other sub-goals, thus creating a tree hierarchy. The AND operator implies the achievement of all decomposing goals are necessary for the target goal to be satisfied. The *means-end* relationship is also used for introducing tasks in the goal hierarchy as the operationalization of goals. Tasks are atomic steps for modifying the current system as a consequence of pattern instantiation. The means-end link indicates that the achievement of the task implies the full satisfaction of the target goal. The whole solution is provided by executing all selected tasks. Some tasks can be connected to resources that represent elements that will be introduced into the system in order to generate the solution. They can be structural elements (attributes, methods, abstract classes, interfaces) or behavioral elements (events, method calls, and so on).

Solution 1: LOCAL	**Solution 2: REMOTE**
Forces: to protect the access to an object, to create objects on demand	Forces: to simulate a local representative for a remote object
Plan: • the *real subject* objects and the *proxy* objects implement the subject interface • each *proxy* object maintains a reference to a *real subject* object • *clients* objects maintain a reference to *subject* objects	Plan: • the *real subject* objects and the *proxy* objects handle a protocol to communicate • *proxy* objects implement the *subject* interface • *clients* objects maintain a reference to *subject* objects

Table 2. Two possible implementing solutions for the Proxy pattern

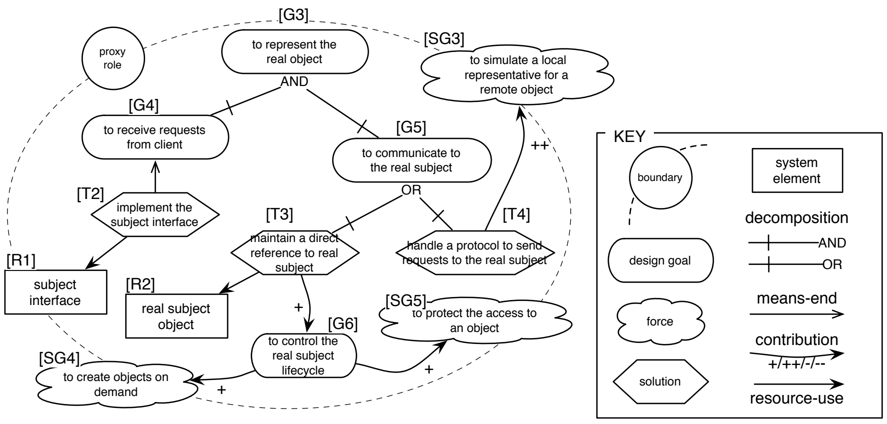

[G3]
to represent the real object
proxy role
[SG3]
to simulate a local representative for a remote object
AND
[G4]
to receive requests from client
[G5]
to communicate to the real subject
++
[T2]
implement the subject interface
OR
[T3]
maintain a direct reference to real subject
[T4]
handle a protocol to send requests to the real subject
[R1]
subject interface
[R2]
real subject object
+
[SG5]
to protect the access to an object
[G6]
to control the real subject lifecycle
[SG4]
to create objects on demand
+
+
KEY
boundary
system element
decomposition
AND
OR
design goal
force
means-end
contribution
+/++/-/--
solution
resource-use

Fig. 2. Goal/Plan diagram for the main role in the Proxy pattern

OR Decomposition. The main strength of this representation is the natural capability to represent alternative paths for the implementing solution. In fact each OR decomposition introduces a decisional point in the goal-model that provides a description of mutually exclusive ways of satisfying a target goal. This alternative is referred to a specific design choice that designer can select in order to customize the solution for the context. This analysis uses contributions for giving the designer an instrument to balance these trade-offs. *Contributions* are a technique for specifying the impact goals and tasks have against context forces, in order to give details of pattern applicability or eventual drawbacks. A (weak/strong) positive impact means the task introduces benefits to the specified force, otherwise a (weak/strong) negative impact indicates a conflict against that force.

Figure **??** shows a slice of the goal/plan model for the Proxy pattern, built on the GoF's book specifications (the client and real-subject roles are omitted). The proxy role is responsible of *[to represent the real object]* (G3). This goal is claimed by the achievement of two sub-goals: *[to receive requests from clients]* (G4) and *[to communicate to the real subject]* (G5). G4 is achieved by the *[implement the subject interface]* (T2) task, that introduces a new interface (R1) used by clients to access to some methods. On the other hand, the communication between proxy and real subject (G5) is possible in two interchanging ways: *[maintain a direct reference to a real subject]* (T3) or *[handle a protocol to send requests to the real subject]* (T4). The solution issue T3 is more suitable for cases in which the proxy controls the real subject life-cycle (to create objects on demand, or to protect the access to the object), whereas solution T4 is required when the real subject is a remote object.

Resolution of Forces. Given this kind of pattern representation, the solution is provided by choosing among design alternatives and then by selecting the corresponding tasks. Design issues have to be balanced in according to the specific application context. Context forces are the means for giving different weight to pattern objectives. The design pattern will solve the context problem whether all main role goals are fully addressed by task selection. AND/OR decompositions, means-end links and contribution links are fundamental to check this property. As shown in the previous example (Figure **??**), by trading with contextual forces, designer can choose al least two different solutions (summarized in Table **??**), one suitable for local proxies and one for remote proxies. This example considers the proxy role only, but actually alternatives in the Proxy pattern are more than the two described before.

4 Approach Analysis and Argumentation

The main idea of this goal-based representation is to focus on the design pattern rationale rather than on the solution structure only. This provides an instrument, complementary to the traditional textual description, that summarizes pattern motivations within all implementing details, thus improving understanding and reuse.

Understandability Issues. The traditional textual description (typically found in the GoF's pattern catalogue [**?**]) includes a lot of information spread across various sections. The comprehension of a design pattern requires a great effort in studying an average of 10 pages for each pattern. This work proposes to replace or to complement the very detailed description with a couple of compact diagrams reporting the most relevant information. The actor diagram provides an explicit structure where intent, applicability and consequences are highlighted. This aids in quickly searching in a catalogue for selecting the best pattern in according to the specific context problem. The explicit reference to intent and applicability clarifies the actual difference among some patterns that present very similar structures; for instance State and Strategy patterns have an identical structure that becomes inexpressive for understanding their totally different intents. Figure **??** highlights their differences. Instead, the couple actor and goal/plan diagrams provides means for deeply understanding the rationale of each implementation detail, and eventually selecting the best alternative solution to apply.

Reuse Issues. The approach provides an useful support for forward engineering and design traceability that is totally independent of the kind of design methodology the user is following up. This means that the approach can be exploited in a traditional design methodology as well as in a goal-oriented one. The approach covers all perspectives of pattern documentation and reuse, from motivation, applicability and consequences to the implementation in a compact and readable form. It is not limited to the final solution to reuse in a specified context, but it also catches the whole reasoning that led to the pattern definition. The explicit documentation of pattern rationale raises the reusability of the pattern, allowing for considering design choices leading to the final desired result. This issue has been already successfully explored by Gross and Yu (in [**?**]) and the proposed approach can be considered complementary to the cited paper. It is worth noting that there is not a direct relationship between functional requirements and pattern goals, because they belong to different domains: design is the activity to realize objectives emerged during the problem analysis, thus requirements give indication about desired functionality, whereas design patterns solve problems of design.

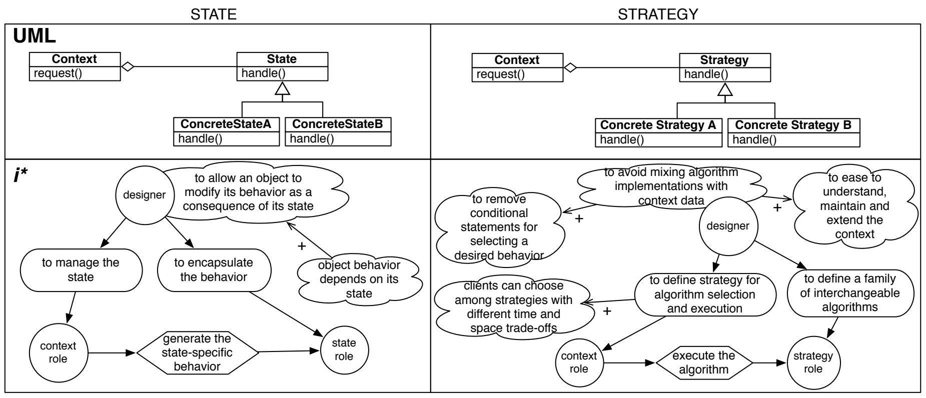

STATE
STRATEGY
UML
Context
request()
State
handle()
ConcreteStateA
handle()
ConcreteStateB
handle()
Context
request()
Strategy
handle()
Concrete Strategy A
handle()
Concrete Strategy B
handle()
i*
designer
to allow an object to modify its behavior as a consequence of its state
to manage the state
to encapsulate the behavior
object behavior depends on its state
+
context role
generate the state-specific behavior
state role
to avoid mixing algorithm implementations with context data
to remove conditional statements for selecting a desired behavior
to ease to understand, maintain and extend the context
designer
+
+
clients can choose among strategies with different time and space trade-offs
to define strategy for algorithm selection and execution
to define a family of interchangeable algorithms
+
context role
execute the algorithm
strategy role

Fig. 3. Comparison of two commons patterns from GoF book that present a very similar UML structure.

5 Related Works

Several works have been proposed for stating design pattern solutions, improving traceability and maintenance issues; such problems become even greater when more patterns are used in composition. Pattern specification languages that utilize mathematical notation provide the needed formality, but often at the expense of usability. Mikkonen in [?] applies rigorous formalization to pattern solution, in a way that reasoning can be made on pattern temporal behaviors in terms of high-level abstractions of communication.
Many approaches, including the original GoF diagrams [?], use a subset of the UML notation for the pattern formalization. UML is very good at communicating designs, and it is also continuously evolving for better expressiveness. Class, sequence and activities diagrams are the most frequently used for representing structure and collaboration views. Rielhe introduces role diagrams [?], using a notion of role that is more abstract. These diagrams define roles played by objects and thus the views objects hold on each other. Rielhe relies on roles mainly to address boundary conditions in recursive structures, explicitly focusing on developing and documenting object collaboration patterns. Sabatucci et al [?] front the problem of design pattern composition, by introducing a fine-grained description of the static and dynamic aspects of a pattern solution. Composition is provided by a small set of operators working on solution elements, tracing transformations before pattern instantiation into the system. They stress the importance of representing the pattern semantics for increasing the consistence and the reusability of multi-patterns.
In [?], the author proposes of considering non-functional requirements (NFR), coming from the analysis phase, during the design. These are treated as design goals, leading the designer to explore a design history of alternative choices, by reasoning on the impact of each pattern over non-functional requirements. An extension of this approach was carried on by Weiss [?] who introduces a rigid form for soft-goal hierarchy to reason about a pattern. This structure is built starting from a standard set of NFR crossed with other NFRs coming from the problem context.

6 Conclusions and Future Works

Some interesting directions can lead to future developments. The first issue concerns the *independence from a domain*. The emphasis given on representing properties and structures of a solution against a specific need makes the approach domain-independent. It is interesting to investigate whether the same approach could be extended to support different categories of patterns, for example analysis patterns, architectural pattern, agent-oriented pattern, aspect-oriented-patterns and so forth.

The second issue concerns the *semantics of the solution*. The nature of this representation is semi-formal, in fact, instructions inside each task are given in natural language, so they mey be unclear, redundant or incomplete. The research question is whether it is possible to identify a formal semantics to describe these tasks. Some directions that will be explored are: (i) to use the meta-modeling for defining an ontology of solutions [**?**] and (ii) the integration of the approach with the aspect oriented programming in order to allow automatic generation of aspectized-design patterns (as proposed in [**?**,**?**]).

Finally, this representation can be extended to include pattern composition operators given the importance to consider reciprocal force influence occurring when multi-patterns are used to solve a conjoined problem [**?**]. Resolution of forces must consider possible conflicts and semantically inconsistent situations. It is interesting to analyze (semi-automatic) reasoning techniques and tools for for identifying these problems.