A Model for Meta-Specification and Cataloging of Software Patterns

León Welicki
Facultad de Informática
Universidad Pontificia de Salamanca,
campus Madrid
c/Sorolla 4, Madrid, Spain
lwelicki@acm.org

Oscar Sanjuán Martinez
Facultad de Informática
Universidad Pontificia de Salamanca,
campus Madrid
c/Sorolla 4, Madrid, Spain
oscar.sanjuan@upsam.net

Juan Manuel Cueva Lovelle
Department of Computer Science
University of Oviedo
Calvo Sotelo s/n, Oviedo, Spain
cueva@lsi.uniovi.es

Abstract

In the last years, the patterns community grew at a very fast pace. A lot of pattern languages and pattern systems emerged. Some of them are very popular, some of them are not. The patterns in each pattern language are described using different templates and the samples are written using different programming languages. This produces a considerable impedance mismatch among all of them. Currently, an abstract and standardized way for specifying software patterns doesn't exist. There isn't any single meta-definition model that governs patterns description. This leads to the problem of finding a way to represent patterns in a self-contained format independent of the programming languages and execution platform. The creation of a language for pattern meta-specification and a catalog of patterns from different pattern languages described using this language is a clear step towards the solution of this problem. Additionally, the creation of a web-based visualization tool for the catalog makes this knowledge available to the world, allowing searching, linking and using the patterns in the catalog. In this paper, we will present the results of our research work on meta-specification and cataloging of software patterns.

Keywords: *patterns, Meta, meta-specification, classification, patterns browser, pattern oriented software engineering*.

1. Motivation

In the last years, the patterns community grew at a very fast pace. A lot of pattern languages and pattern systems emerged. Some of them are very popular, some of them are not. To show the tip of the iceberg, we propose a very simple, yet mind provoking, exercise: let's sum the number of patterns in 3 reference and mainstream books, "Design Patterns" [GoF95], "Pattern Oriented Software Architecture, Volume 1" [POSA96] and "Patterns of Enterprise Application Architecture" [Fowler03]. The final number is 100… quite big to remember all! To worsen things, each pattern language is described using different formal elements (templates) and the samples are written with different programming languages. This produces a considerable impedance mismatch among all of them. There is not a single model to guide their definition. Such a guidance model may not be restrictive: the existence of a set of rules doesn't imply fewer possibilities or more difficulties of representation. In fact, we can take as an example the C programming language: the same program can be written in almost infinite ways and in all cases will be a set of instructions understandable by someone or something, in this case, a computer.

Copyright © 2005 León Welicki. Permission is granted to copy for the PLoP 2005 conference. All other rights reserved.

Additionally, we found some "calls to action" in several reference books in the patterns literature that made evident the need for a standardized way to describe and store patterns and a simple way to use these descriptions. This makes patterns meta-specification a first class problem that must be solved and hasn't been successfully addressed yet. In the next section, we'll quickly walk through these "calls to action"…

1.1 Calls to action

The clearest use case and "call to action" for the development of a meta-specification language and a cataloging tool can be found in the last chapter of POSA Volume 1 [POSA96], where the authors refer to the future of patterns:

> *…In conclusion, using patterns successfully still requires the intellectual skills of the software developer. We believe that a well-designed pattern browser or World Wide Web tool can be much more efficient in helping a developer to find and use patterns that a fully integrated "pattern-supporting" software development environment ever could be. [POSA96].*

Another statement that illustrates difficulties of patterns representation is the one found in the GoF book [GoF95]:

> *Finding patterns is easier than describing them [GoF95]*

Steve Berczuk stated that:

> *Alexander's pattern language contains over 250 patterns, organized from high level to low level. The goal in documenting patterns that exist in software architectures is to arrive to a similar system, but this will take time [Berczuk94]*

The creation of a patterns catalog from different languages and system of patterns (GoF, POSA, Patterns of Enterprise Application Architecture (PEAA) [Fowler03], etc.) specified using the same syntactic and semantic constructs, with relational capabilities is a clear step towards this direction.

> *…look for patterns you use, and write them down. Make them a part of your documentation. Show them to other people. You don't have to be in a research lab to find patterns. In fact, finding relevant patterns is nearly impossible if you don't have practical experience. Feel free to write your own catalogue of patterns...but make sure someone else helps you beat them into shape! [GoF95]*

Writing patterns is a very important task for leveraging knowledge within an organization or in the software engineering community as a whole. A set of tools like this could simplify the pattern mining and writing tasks, enhancing the efficiency in knowledge sharing.

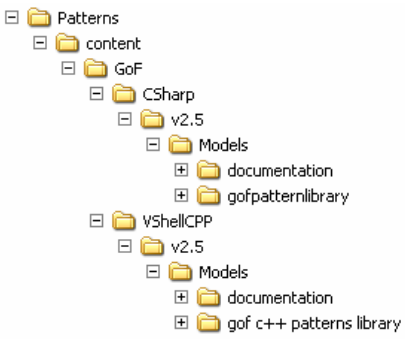
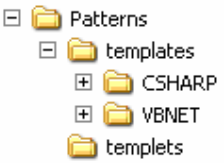
2. Context

Most modern modeling tools have pattern facilities that allow navigating a catalog of patterns and generating software artifacts out of them. All of them have some drawbacks and limitations regarding our original problem – specifying patterns in an abstract format. In this section we will analyze some modeling tools and some Web-based pattern catalogs showing how all of them lack an abstract and standardized way for specifying patterns.

2.1 Patterns in commercial and research tools

Although for some people the idea may sound new, lots of efforts on this area had been done in the last years by modeling tools vendors, by the academic community, and by some research organizations.

One of the most inspiring works is captured in the article "Code generation from design patterns" [BFVY96]. In this research work, some researchers at IBM created a web based catalog for the GoF patterns with code generation facilities. Similarly, almost every modeling tool in the market has some pattern-related functionality. In all cases, there is a clear lack of an abstract meta-model approach to guide patterns definition. In the following table, some commercial and research tools are briefly analyzed.

Tool	**How Patterns are represented…**
Rational XDE	Patterns are specified using a combination of XML, XSD and HTML files. For each programming language-specific pattern implementation a set of files must be created. The most important file is an XML document (generally its name coincides with the pattern name) and has all the necessary information to use the pattern within XDE. In the following figure, the folders structure of the XDE patterns files (for the .NET version) is shown. Notice that there is a hierarchy for each target implementation language (C#, C++, etc.). 
Together	Patterns are specified using a set of XML files with a proprietary format. For each programming language-specific implementation a set of files must be created. The next figure shows the folders structure for the .NET version. 

Enterprise Architect	Patterns are specified in an XML file, using XMI. The pattern definition does not include full information of the knowledge level nor algorithm implementations (implementation level). It only contains structural information.
IBM Research Prototype	The "knowledge" level of the patterns is represented as a set of HTML files, where each one represents a section of the pattern template (using the GoF template). Therefore, 13 HTML files are necessary to represent the literary level of a pattern. The relationship between them is only syntactic. The source code (implementation level) is described using COGENT (specially created for this project). It is not language independent (is necessary a template for each language) and is not related with the knowledge level.
Pattern By Example	Patterns are described in a single XML file. A special section for each target language must be created. It is mainly targeted to patterns that have an implementation level.

Table 1 – Analysis of some commercial and research tools with patterns support.

There is a common factor in all the tools: none of them has a unified and abstract way for representing patterns in the way proposed at the beginning of this work. In all cases, separate information exists for each target programming language and environment. Additionally, in some cases literary information (knowledge level) is not present or is not related to the implementation level. A detail analysis of each of these tools can be found in [Welicki04].

2.2 Some existing Web-based catalogs

It is unfair to say that no web-based patterns catalog exists today. In fact, there are many of them and some are very good. As an example, we can name PatternShare [Microsoft04], a Microsoft initiative led by Ward Cunningham.

Other available catalogs are the Portland Pattern Repository, Sun's J2EE Blueprints, The Server Side Patterns, Enterprise Integration Patterns Catalog, UI Patterns and techniques, Implementing Finite State Machines, etc.

In all the tools mentioned above we have found occurrences of some of this problems: patterns have some dependence on a particular platform, the implementation level is not described in an abstract format, the navigability is not always very clear, search facilities are not always present or usable, and most important of all, there is no a single meta-definition model that rules the patterns description.

3. Problem

How can we represent patterns in a self-contained format independent of the programming languages and execution platform?

3.1. Problem restated: establishing objectives

After establishing our problem and contrasting it with our motivations and context, we could clearly establish the objectives of our research:

- Represent patterns in a format independent of the programming languages and execution platform. This representation must be self-contained, leveraging the knowledge and implementation levels.

- The creation of a catalog of patterns represented with that format and the creation of a tool to navigate that catalog. The tool must be easily accessible for a broad audience.

4. Our Proposed Plan

To solve the meta-specification and classification problem, we propose the following steps:

- Create a meta-specification language to represent the patterns. This meta-specification language must be able to represent the knowledge and implementation levels. The definition of the implementation level must be independent of any programming language and implementation platform.

- Create a catalog of patterns described with the language created in the previous step

- Create a tool to browse a catalog of patterns described with the language mentioned above. Ideally, this tool must be based on Web technologies, to make it accessible to a broad audience through a web browser (without the need of any additional software). Additionally, it may expose some of its functionalities as Web services which may allow using the catalog in different ways (for example, it could be invoked from development IDEs like Eclipse, Visual Studio .NET, etc.).

4.1 Patterns redefined

It is very difficult to establish a definition of the term "pattern". There are several slightly different definitions in the reference books that exist in this field. In order to create our meta-language, having a clear definition of this term is mandatory.

A very inspiring definition is the one that can be fount at [BFVY96]. This definition could be taken as a "proposal declaration" for pattern-oriented CASE tools:

> *A design pattern only describes a solution to a particular design problem; it is not itself code. Some developers have found it difficult to make the leap from the pattern description to a particular implementation, even though the pattern includes code fragments in the Sample Code section. Others have no trouble translating the pattern into code, but they still find it a chore, especially when they have to do it repeatedly. A design change might require substantial reimplementation, because different design choices in the pattern can lead to vastly different code.*

For the purpose of this research work, we will define a pattern as a "*piece of knowledge that includes information about a problem and its solution in a specific context, with the trade-offs and all the literary information needed to have a good understanding of the issues related with it. Eventually, it may contain implementation specific information,*

which will allow generating the source code for the proposed solution to the problem". This is shown in figure 1.

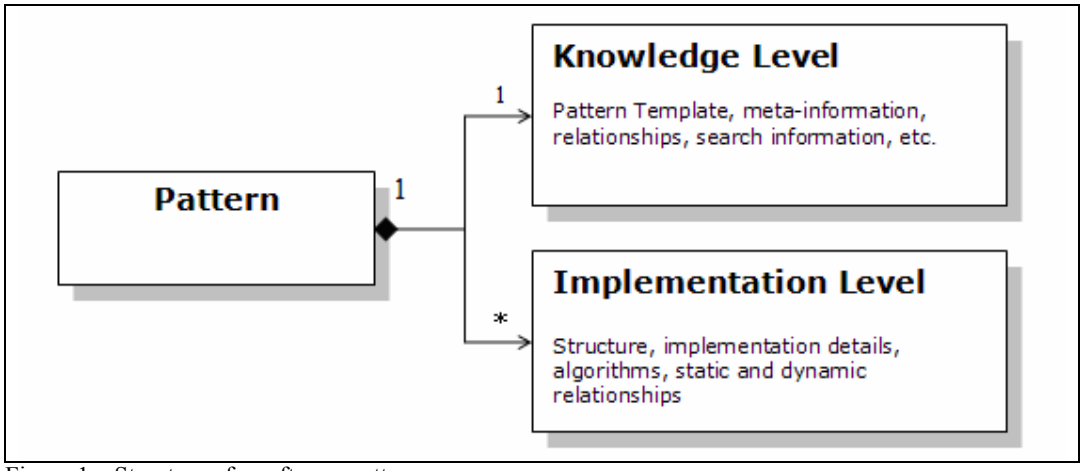


Figure 1 – Structure of a software pattern.

5. Requirements

5.1 Requirements for the meta-specification language

The main requirement for the solution is to establish a uniform and standardized way to represent patterns. This pattern definition language must provide appropriate syntactic and semantic constructs to represent the knowledge and implementation levels in a homogeneous way.

Regarding the implementation level, there shouldn't be special clauses to describe platform or language specific semantics. The behavior of the participants of the patterns must be described as abstract as possible.

Our meta-language should satisfy the following requirements:

1. **Uniformity:** use a uniform language to describe the knowledge and implementation levels.
2. **Abstraction:** the representation at the implementation level must be as abstract as possible.
3. **Platform independence:** implementation description must be totally independent of the execution platform and programming languages.
4. **Support for patterns with and without implementation**: it must be able to represent equally patterns with knowledge and implementation levels (e.g. GoF patterns [GoF95]) and the ones that only have the knowledge level (e.g. Analysis Patterns [Fowler97]).
5. **Intelligibility:** must be easy to understand by humans.
6. **Relational:** the language must support relationships among patterns. Navigability between patterns must be easy and straightforward.
7. **Easy production:** it must be easy to write patterns using this language. Patterns should be able to be written and modified using any text editor, like VI or Notepad.

8. **Results tuning:** implementation level output must have tuning facilities: patterns are a starting point. New solutions and use cases for a particular pattern can emerge by adjusting the implementations details and trade-offs
9. **Leverage existing knowledge:** it must provide the possibility to add cross references to further sources of information on a particular pattern (like external web sites or books), allowing the reuse of the existing knowledge at a global scale. It must also provide the basis to gather information on a pattern from different sources.
10. **Metadata support**: it must have metadata for searches. Ideally, patterns could be annotated with Resource Description Framework (RDF) [RDF] or Ontology Web Language (OWL) [OWL].
11. **Interoperability/interchangeability:** it must provide information exchange facilities like XMI support. This will allow integrating the patterns described using this language with existing CASE tools.

5.2 Requirements for the patterns catalog

The patterns catalog gathers the meta-specifications of the patterns created using the language presented in the previous section. It must support the possibility to easily add, remove, modify and link patterns.

The patterns catalog should satisfy the following requirements:

1. **Easy registration**: pattern meta-definitions must be easily added, removed, modified and linked.
2. **Easy to manage**: it must be easy to administrate.
3. **Expose patterns through a SOA Interface:** expose information through a service-based interface (following SOA guidelines)
4. **Search Facilities:** the catalog should provide search facilities which could leverage the metadata on the meta-definition language or use simple pattern matching, taxonomies, ontologies, etc.

5.3 Requirements for the catalog browser

The catalog browser is the component that shows the contents of the catalog and allows the user browsing and using the patterns.

Following, its main requisites are listed:

1. **Multiplatform:** it must be enabled to be used in heterogeneous platforms
2. **Accessible through Web-based Interface:** it must be available through a web-based user interface.
3. **Usability:** easy to use and intuitive
4. **Multiple Views on a Pattern:** expose multiple views on each pattern in catalog. As an example, we could establish that GoF patterns have a complete view, a summary view, a CRC cards view and a source code view. The views should support personalization on a pattern basis.
5. **Navigability:** provide easy and efficient navigation through the patterns in the catalog. Make the most of the relations between patterns.

6. Implementation

Through our research, we partially solved the meta-specification problem. We built a research prototype composed of a patterns meta-specification language, a catalog of patterns described using that language and a web-based patterns browser.

The elements that shape this solution are the ones as follows:

- **M4PS (Meta-language for Pattern Specification)**: M4PS is a pattern meta-specification language based on XML. It allows defining a pattern in single self-contained file. It includes language constructs that allow the abstract representation of the knowledge and implementation levels. The implementation level is described independent of the target language and platform.
- **Patterns Catalog**: is a catalog of representative patterns, written using M4PS
- **Catalog Browser:** is a visualization and navigation tool for the catalog. It supports multiple views on a pattern and exposes the catalog through a SOA interface (based on SOAP Web services).

In the next figure (figure 2), the stack of technologies developed for the solution is shown graphically.

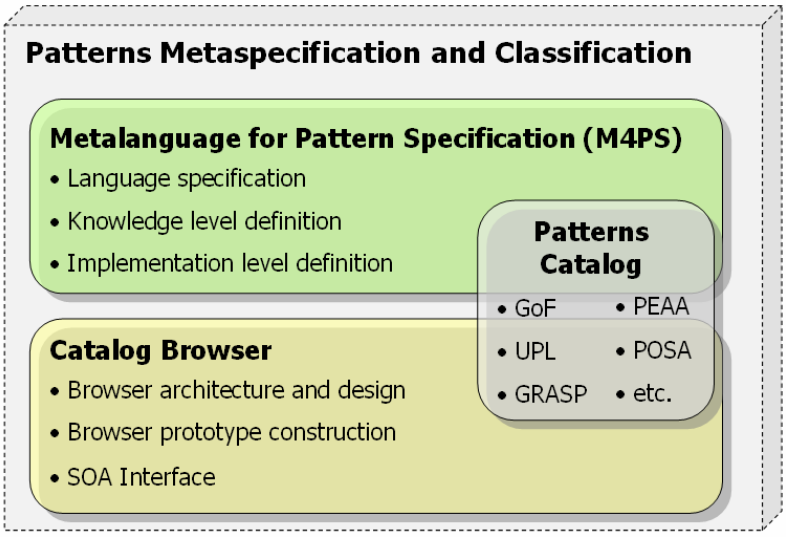


Figure 2 – Technologies created during this research work.

In the remaining part of this section we will explore each one of the building blocks that conforms the solution. The current version of the solution fulfills a great number of the requisites established in the section 5 of this paper, but not all of them.

6.1 M4PS Meta-language

M4PS is the acronym for Meta-language for Pattern Specification. M4PS is a meta-language created with the purpose of describing all kind of software patterns. These patterns must have literary information (knowledge level) and may also have behavior information (implementation level). In its first version (0.9.3, which is the current

version) it supports almost every requirement previously defined (see section 5.2), except for XMI, RDF and OWL support (requirements 10, and 11 respectively). M4PS is based on XML.

For the description of the implementation level, a Domain Specific Language (DSL) containing the representation of abstract constructions of object oriented programming languages has been created. The full definition of this subset of the language can be found at [Welicki04].

6.1.1 Representing a pattern in M4PS

M4PS allows representing a pattern in a single file. This representation could be used to view the literary information, view CRC cards from the pattern or generate code in any language (in the first version, C#, Java and VB.NET code generators are provided).

An M4PS pattern representation is self-contained: this means that it contains all the literary information needed to understand the pattern, all the implementation information to use the pattern (to allow code generation from it or using it in a case tool) and metadata to assist the search engine and the catalog browser.

Patterns are represented in M4PS through 4 sections:

- **Meta**: includes meta-information about a pattern. It is used mainly for search and contextual browsing.
- **Template:** literary information of the pattern.
- **Structure**: structure of the pattern (participants with their relationships and responsibilities).
- **Implementation**: implementation information of the pattern (implementation level).

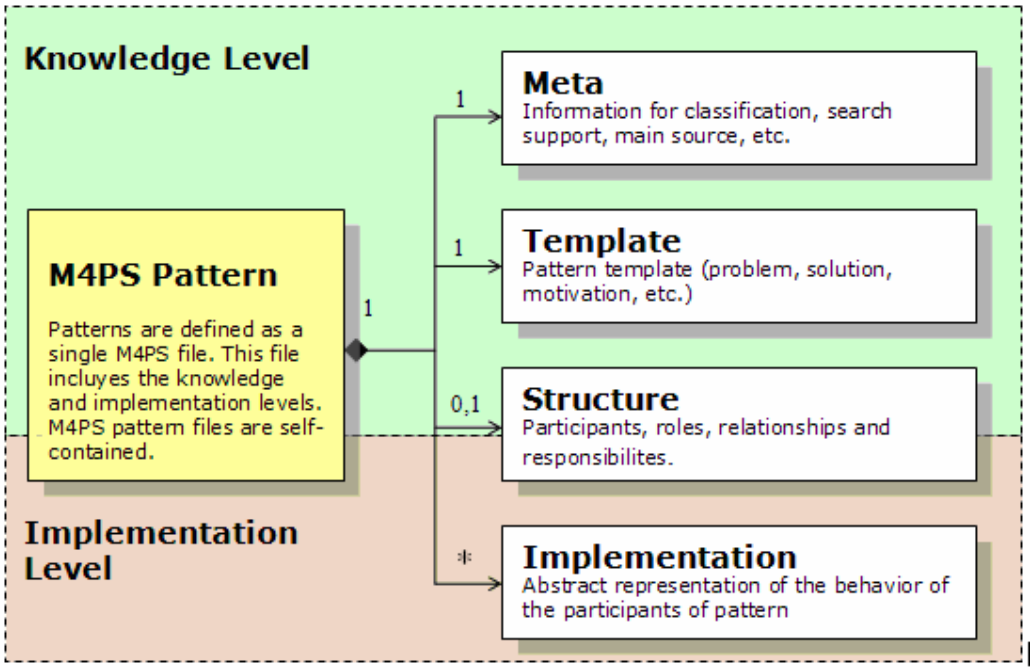


Figure 3 – Defining patterns in M4PS.

As we can see on figure 3, the knowledge level consists of the Meta, Template and Structure sections. The implementation level consists of the Structure and Implementation sections. As we've seen in the figure 3 above, Structure section is shared by the knowledge and the implementation levels, because it contains a formal enumeration of the participants and their relationships (implementation level) and textual information about their responsibilities (knowledge level).

It is important to notice that the Structure and Implementation sections may be optional. Some patterns may not need them. For example, the patterns at "A UML Pattern Language" [Evitts99] may not have an implementation section. The Anti Patterns [BMMM98] may have both structure and implementation as well. The design patterns from GoF [GoF95] have both knowledge and implementation sections. As a conclusion, we can say that the knowledge level is always mandatory and the implementation is optional.

6.1.2 Describing the Knowledge Level

Describing the knowledge level is a very complex task. A big number of pattern templates exist and creating a new template that summarizes all of them is not a trivial task. In order to create a template for the knowledge level, we analyzed the most popular ones (including the GoF Template, Analysis Patterns Template, POSA, Patterns of Enterprise Application Architecture, Alexander Form, Canonical Form, etc.).

In the first version of our prototype, we created a flexible template that combines elements of the Canonical Form, the GoF Template, POSA and the Alexandrian form. We are actually working on a more powerful and flexible approach based on semantic definition elements.

6.1.3 Describing the Implementation Level

M4PS contains a set of constructs that allows describing the behavior of a set of software artifacts without using any particular programming language and without any platform compromise.

- Artifact description: describing code modules (e.g. classes) and its elements
- Algorithm description: describing the behavior of a code module function (e.g. methods)

M4PS can be translated to any high-level programming language. In the first version of the solution prototype, 3 translators are provided: C#, VB.NET and Java.

A full definition of the implementation level constructs can be found in [Welicki04].

6.2 The Patterns Catalog

The patterns catalog has been totally built using M4PS. In its first version, it includes some patterns from the GoF book, some others from "A UML Pattern Language" and some from GRASP [Larman99].

The catalog is a set of files, where each one contains the definition of a pattern in M4PS (each M4PS represents a pattern in a totally self-contained fashion). There are other XML files that have information on the categories and sources of information. All this information is not yet included in the M4PS specification, but it will in the next version.

Additionally, there is a registry file where all the patterns are "registered" in the catalog. This means that is not sufficient to write a M4PS pattern file on disk. It must be also "registered" in the patterns registry, which is also an XML file.

The catalog is maybe the most complex and important part of out prototype.

6.3 The Patterns Browser

The patterns browser is a web-based tool that allows browsing through the contents of the catalog. The objective of the pattern browser prototype is to provide a viewing application for the M4PS patterns catalog.

6.3.1 Views on a pattern

The pattern browser allows navigating through the pattern languages registered in the system. For each pattern, there are several views available. Therefore, taking a pattern as the model (in a MVC like architecture), multiple views can be applied to it (as shown in figure 4). Moreover, the views can be dynamically attached to a pattern, allowing changing the visualization style for different patterns based on arbitrary criteria.

In the current version, 5 views are provided for a pattern:

- **Complete View**: shows all information on a pattern (taken from the template)
- **Summary View**: a summary of the knowledge level of the pattern
- **CRC View**: shows CRC cards for the participants of the pattern
- **Source Code View**: shows the source code for the pattern. Several implementation options can be chosen.
- **M4PS View**: shows the M4PS code for the pattern

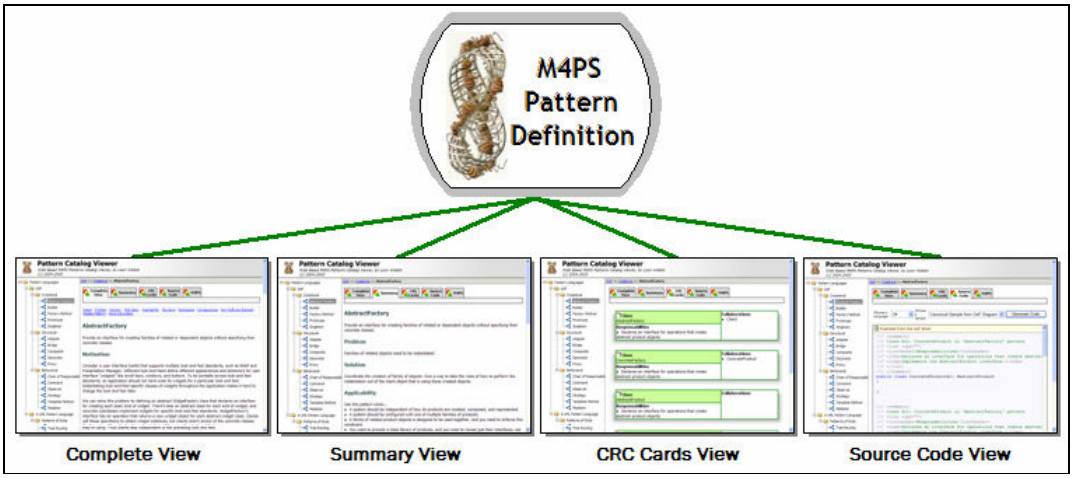


Figure 4 – Multiple views on a pattern. New views can be attached to any pattern in the catalog.

The patterns browser fulfills all the requirements established previously in section 5.3.

6.3.2 Usability

Usability should be a first class requirement in any Web-based application that exposes a GUI. One of our design objectives was to provide an easy to use and intuitive user interface to the meta-specifications catalog. Therefore, for the design of the user interface, we followed usability principles from [Nielsen99] and [Krug99] (among others). A multi-dimensional model for text representation approach like the one presented in [Welicki04c] is currently under consideration.

6.4 Full-Solution Prototype High Level Architecture

The model prototype has been developed as a distributed application, using layers architectural pattern [POSA96]. There are 3 layers following a very classical categorization (presentation, business and data). Each tier is as autonomous and loosely coupled as possible. Figure 5 shows a brief architecture model for the solution. The prototype implementation has been built using .NET technologies (ASP.NET, C#, etc.).

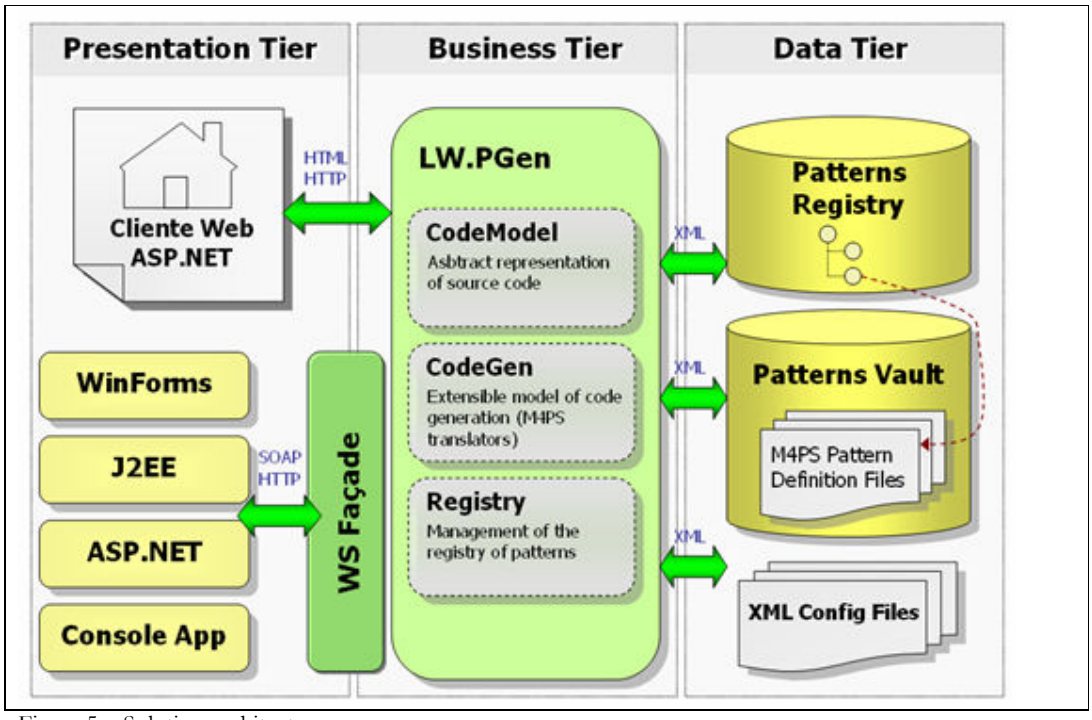


Figure 5 – Solution architecture.

6.5 Describing a pattern using M4PS: Case Study

In this section we will show briefly how to describe patterns using M4PS. A full M4PS description of a pattern is very long. Therefore, we selected some meaningful examples of each of the sections of the pattern's definition, so the reader can have an accurate idea of how a M4PS looks like.

6.5.1 Describing the metadata

Patterns are annotated with some contextual metadata, which can be used for searching or inference engines. The following fragment is a partial view of the metadata for the Singleton [GoF95] pattern:

```
<meta>
   <friendlyName>Singleton</friendlyName>
   <category>GoF.Creational</category>
   <source>GoFBook</source>
   <author>GoF</author>
   <scope>Object</scope>
   <purpose>Object Creation</purpose>
   <context>Development</context>
...
... Rest of the meta section elided for simplicity
...
</meta>
```

Snippet 1 – "Meta" section in Singleton's M4PS pattern definition.

6.5.2 Describing the literary information

The knowledge level (or literary level) is the largest part of the pattern. The following snippet illustrates the knowledge level for the Abstract Factory [GoF95] pattern:

```
<template>
   <intent>Provide an interface for creating families of related or dependent
objects without specifying their concrete classes</intent>
   <problem>Families of related objects need to be instantiated</problem>
   <solution>Coordinate the creation of family of objects. Give a way to take
the rules of how to perform the instantiation out of the client object that is
using these created objects.</solution>
...
... Rest of the template elided for simplicity
...
</template>
```

Snippet 2 – "Template" section in Abstract Factory´s M4PS pattern definition.

6.5.3 Describing the structure

The structure section contains the structural information of the pattern. It is divided in three parts:

- **Participants**: lists the participants of the pattern
- **Relationships**: relationships between the participants. The relationship types are inheritance, composition, aggregation, association and creation.
- **Responsibilities**: textual definition of the responsibilities of each participant

The next snippet shows the structure section for the Abstract Factory pattern:

```
<structure>
   <participants>
      <participant role="AbstractFactory" isAbstract="true"  cardinality="1"/>
      <participant role="ConcreteFactory" isAbstract="false" cardinality="+"/>
      <participant role="AbstractProduct" isAbstract="false" cardinality="+"/>
      <participant role="ConcreteProduct" isAbstract="false" cardinality="+"/>
      <participant role="Client"          isAbstract="false" cardinality="1"/>
   </participants>

   <relationships>
      <relationship type="inheritance"
             child="ConcreteFactory" parent="AbstractFactory"/>
      <relationship type="inheritance"
             child="ConcreteProduct" parent="AbstractProduct"/>
      <relationship type="creation"
             source="ConcreteFactory" target="ConcreteProduct"
             cardinality="1"/>
      <relationship type="association"
             source="Client" target="AbstractFactory"
             cardinality="1"/>
   </relationships>

   <responsibilities>
      <role name="AbstractFactory">
        <responsibility>Declares an interface for operations that create
abstract product objects. </responsibility>
      </role>
      <role name="ConcreteFactory">
        <responsibility>Declares an interface for operations that create
abstract product objects.</responsibility>
      </role>
      <role name="AbstractProduct">
        <responsibility>Declares an interface for operations that create
abstract product objects.</responsibility>
      </role>
      <role name="ConcreteProduct">
        <responsibility>Declares an interface for operations that create
abstract product objects.</responsibility>
        <responsibility>Implements the AbstractProduct
interface.</responsibility>
      </role>
      <role name="Client">
        <responsibility>Uses only interfaces declared by AbstractFactory and
AbstractProduct classes.</responsibility>
      </role>
   </ responsibilities >
</structure>
```

Snippet 3 – "Structure" section in Abstract Factory´s M4PS pattern definition.

This structure makes it very easy to port patterns to M4PS because it allows focusing on each individual aspect of the structure separately. The user first identifies the participants and then establishes the relationships between them. Finally, he adds the responsibilities to each role.

6.5.4 Describing the implementation level

As we stated previously, M4PS has a DSL to represent source code constructs at a high level of abstraction. Patterns that specify an implementation must have a "baseImplementation" section, which describes each participant and its basic implementation. A participant description contains three sections for describing its implementation: Properties, Constructors and Methods.

In the next snippet we will show how to define the base implementation section for the Singleton pattern:

```
<baseImplementation>
   <participant role="Singleton">
      <properties>
         <member scope="private,protected" name="instance"
                 type="Singleton" isClass="true"/>
      </properties>
      <constructors>
         <constructor>
            <signature scope="private,protected"/>
         </constructor>
      </constructors>
      <behavior>
         <method name="getInstance">
            <signature scope="public" returns="Singleton"
                       isClass="true" isAbstract="false"/>
            <implementation>
               <if>
                  <condition>
                     <isNull variable="instance"/>
                  </condition>
                  <truePart>
                     <createInstance variable="instance" type="Singleton"/>
                  </truePart>
               </if>
               <return variable="instance"/>
            </implementation>
         </method>
      </behavior>
   </participant>
</baseImplementation>
```

Snippet 4 – "baseImplementation" section in Singleton's M4PS pattern definition. The sample above represents the canonical implementation of the Singleton from the GoF book.

The basic implementation can be redefined, allowing a pattern having more than one implementation. We can add additional implementations to a pattern using the "additionalImplementations" section. In this section, we can redefine or augment any of the participants defined in the "baseImplementation" section. Each particular implementation is described inside an "implementation" section.

In the next snippet we show how to add a "Double Check Lock" implementation to our Singleton pattern.

```
<additionalImplementations>
   <implementation name="Double Check Lock">
      <implementationInfo>
         <friendlyName>Double Check Lock Singleton</friendlyName>
         <description>Implementation of the Singleton using the Double Check
Lock idiom</description>
      </implementationInfo>
      <instanceOf role="Singleton" name="SingletonMulti">
         <properties>
            <member scope="private" name="padlock"
                    type="object" isClass="true"/>
         </properties>
         <behavior>
            <method name="getInstance" is="getInstance">
               <signature scope="public" returns="SingletonMulti"
                          isClass="true" isAbstract="false"/>
               <implementation>
                  <if>
                     <condition>
                        <isNull variable="instance"/>
                     </condition>
                     <truePart>
                        <lock on="padlock">
                           <if>
                              <condition>
                                 <isNull variable="instance"/>
                              </condition>
                              <truePart>
                                 <createInstance variable="instance"
                                                 type="SingletonMulti"/>
                              </truePart>
                           </if>
                        </lock>
                     </truePart>
                  </if>
                  <return variable="instance"/>
               </implementation>
            </method>
         </behavior>
      </instanceOf>
   </implementation>
</additionalImplementations>
```

Snippet 4 – "additionalImplementation" section in Singleton's M4PS pattern definition. The sample above represents the Singleton combined with Dual Check Lock.

6.5.4 Describing a full pattern in M4PS

In the previous section we have seen how to represent individual sections of a pattern using M4PS. A full pattern is represented in a single file combining all of the sections presented previously.

Therefore, an M4PS pattern definition has the following structure:

1. Meta
    a. Information for search and inference engines
2. Template
    a. All the knowledge level information of the pattern
        i. Problem
        ii. Solution
        iii. Motivation
        iv. Consequences
        v. Etc.
3. Structure *(optional)*
    a. Participants
    b. Relationships
    c. Responsibilities
4. Implementation *(optional)*
    a. Base Implementation
    b. Additional Implementations
        i. Implementation

7. Conclusion: Benefits and Contributions

The creation of a meta-language for pattern specification and a catalog of patterns from different pattern languages described with this language (using the same syntactic and semantic constructs) is a clear step towards the solution of the stated problem. The creation of a web-based catalog visualization tool makes this knowledge available to the world, allowing searching, linking and using the patterns in the catalog.

Following, the main benefits and contributions of this research are listed:

- A meta-language to describe patterns at an abstract level
    - Patterns from different pattern languages can be described using the same semantic and syntactic elements
    - Behavior description is platform and programming-language independent

- A catalog of patterns, to establish the foundations of practical knowledge gathered from previous experiences
    - Unique source of knowledge on diverse pattern languages and pattern systems
    - All patterns are described using the same semantic and syntactic rules
    - Search facilities
    - Relational facilities. Possibility to link patterns in different pattern languages
    - Built-in support for growing and evolution

- A Web-enabled catalog browsing tool
    - Point of access to a unified repository of knowledge
    - Provide access to the catalog through a simple, accessible and easy to use.
    - Low cost of distribution

8. Work in Progress and Future Work

As we stated previously, our research work does not fulfill all the requirements established in section 5 of this paper. We are still working enhancing the M4PS itself and our prototype of the catalog.

Even though the prototype is ambitious and implements advanced properties it doesn't implement the whole model. It is a conceptual prototype, which implements a great number of all the requisites established previously. The main objective of the prototype was to prove and validate empirically the model.

At the moment of this writing, we are working on the following enhancements:

- **Describing the knowledge level (section 6.1.2)**: We are actually working on a more powerful and flexible approach based on semantic definition elements. We are considering creating an ontology model for describing the template or a semantically annotated template, but that work is currently in progress and is not included in the current version.
- **Patterns Catalog (section 6.2)**: Include constructs in M4PS for describing pattern languages, pattern categories, design principles, etc.
- **Classification:** we are currently working on an advanced classification mechanism for patterns and concepts (principles, refactorings, etc.) to assist cataloging and searching.
- **SOA interface:** in the first version of the prototype, we created a "proof of concept" SOA interface for pattern languages. That interface is not acceptable in a production system. Therefore, we are working on a better interface, so we can make this information accessible to any client.
- **Improve M4PS expressiveness for implementation definition**: M4PS can be further improved to be more expressive. This would imply supporting more data types, more programming statements, etc**.**
- **M4PS validation**: create an XSD (XML Schema) to validate M4PS pattern definitions and to formalize its usage.
- **Search**: current search capabilities are very basic. We are working on advanced searches with inference facilities.
- **Production Qualities:** give the prototype production qualities. This makes the step between an academic "proof of concept" prototype and an industrial application. This implies being robust, auditable, flexible, etc.

References

[Alexander79] Alexander, Christopher. *A Pattern Language*. Oxford University Press, 1979

[Berczuk94] Berczuk, Steve. *Finding Solutions through Pattern Languages*. Computer 27 No 12, December 1994

[BFVY96] Budinsky; Finnie; Vlissides; Yu. *Automatic Code Generation from Design Patterns*. IBM Systems Journal, Vol. 35, No. 2, 1996

[BMMM98] Brown, W., Malveau, R., Mc Cormick III, H., Mowbray, T. *Antipatterns: Refactoring Software, Architectures and Project in Crisis*. Wiley and Sons, 1998.

[POSA96] Buschmann, Frank; Regine Meunier, Hans Rohnert, Peter Sommerlad, and Michael Stal. *Pattern-Oriented Software Architecture, Volume 1: A System of Patterns*. John Wiley & Sons Ltd, 1996.

[Borland04] Borland. *Together Technologies*. http://www.borland.com/

[C2PatternForm04] C2 WikiWikiWeb. *Patterns Form*. http://c2.com/cgi/wiki?PatternForms

[Delta04] Delta Software Techonlogy. *Pattern By Example*. 2004. http://www.d-s-t-g.com/neu/pages/pageseng/et/common/techn_pbe_frmset.htm

[Evitts99] Evitts, Paul. *A UML Pattern Language*. MTP, 1999.

[Fowler97] Fowler, Martin. *Analysis Patterns: Reusable Object Models*. Addison-Wesley. 1997.

[Fowler99] Fowler, Martin. *Refactoring: Improving the Design of Existing Code*. Addison-Wesley. 1999.

[Fowler03] Fowler, Martin. *Patterns of Enterprise Application Architecture*. Addison-Wesley, 2003.

[GoF95] Gamma, Erich; Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[Hillside03] Hillside Group. *Home of the Patterns Library*. 2003. http://hillside.net/

[IBM04] IBM. *Rational XDE FAQ*, 2004. http://www-128.ibm.com/developerworks/rational/library/4304.html#N10403

[Kerievsky04] Kerievsky, Joshua. *Refactoring to Patterns*. Addison-Wesley. 2004.

[Larman99] Larman, Craig. *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development*. Prentice Hall, 1999.

[Microsoft04] Microsoft Corporation: *PatternShare*. http://patternshare.org/

[OWL] W3C: *OWL Web Ontology Language Reference*, 2004, http://www.w3.org/TR/owl-ref/

[PPR04] Portland Pattern Repository http://c2.com/ppr/

[RDF] W3C: *Resource Description Framework (RDF)*, http://www.w3.org/RDF/

[Sparx03] Sparx Systems. *Enterprise Architect Documentation*. 2003.

[Welicki04] Welicki, León. *Meta-especificación y catalogación de Patrones de Software*. MsCs Thesis, Universidad Pontificia de Salamanca, campus de Madrid, 2004.

[Welicki04b] Welicki, León. *Arquitecturas de Software Agiles, Flexibles y Robustas para Ingeniería Web*. PhD Guided Research and Thesis Proposal, Universidad Pontificia de Salamanca, campus de Madrid, 2004.

[Welicki04c] Welicki, León. *XText: Un modelo para creación y visualización de textos en múltiples dispositivos*. Proceedings of the 1st JPIII, Salamanca, Spain, 2004.