International Journal of Computer Science & Information Technology (IJCSIT) Vol 3, No 5, Oct 2011

Design Pattern Management System: A Support Tool Based on Design Patterns Applicability

Zakaria Moudam, Radouan Belhissi, Noureddine Chenfour

Computer Science Department Faculty of Sciences Dhar Mehraz Fez
University Sidi Mohammed Ben Abdellah Fez, 30000, Morocco

Email addresses: zmoudam@yahoo.fr (Zakaria Moudam),

rbelhissi@yahoo.fr (Radouan Belhissi), chenfour@yahoo.f (Noureddine Chenfour)

Abstract

The use of Design Patterns becomes a necessity in software development. Numbers of Design Patterns have been discovered since the "Gang of Four" (GoF) has published their book. The choice between the amounts of these patterns is becoming increasingly embarrassing. Hence the need for tools that helps in choosing and selecting suitable patterns to solve a specific problem becomes imperative. In fact, that must aim the textual part of the description of the Design Patterns, mainly the applicability part. Many current works focus on the structured part while formalizing Design Patterns.

In this paper we present a Data Management System of the Design Patterns of the GoF. We use XML technology to initiate a modeling language for Design Patterns. We present an automated tool support for pattern search: a tool to catalog and search for Design Patterns according to their applicability conditions that would automate query-based search for applicable patterns.

Keywords: Design patterns, XML, Xpath, Xquery, Keyword.

1. Introduction

Since then, several studies on Design Patterns emerged dealing with discovering, developing and implementing new Design Patterns [14], [6], aiming the detection, the instantiation and the application of Design Patterns in software development [9], [2], or handling the classification, the categorization or the formalization of these Design Patterns [18], [16], [11], [24]; Others automate code generation from Design Patterns [1], [7].

Existing approaches to formalize Design Patterns generally cover only the formal description of the solution structure of Design Patterns. While the structure of a Design Pattern explains how it is applied in software design, it does not explain when to apply a Design Pattern for a given design problem. With the great amount of Design Patterns discovered the need for a good structure becomes more pressing. Tichy lists over 100 patterns in his catalogue

DOI : 10.5121/ijcsit.2011.3504

[19] and successes to create a structure for classifying patterns. However, Tichy's classifications are not of much use for understanding implementation similarities of patterns.

Some authors [15] use the intent section of a Design Pattern description to formalize them. However, software tools based on such formalization could enable users to query for a Design Pattern by giving a description of their design problem based on terminology defined in the specification. We think that, not only the intent part will play in the description of a Design Pattern, but other criteria may help to understand and choose the appropriate Design Pattern for a given problem. In other side, new works appears handling keyword-based searching in relational databases [5]. Keyword queries have become a popular alternative to structured query languages. Keyword-based searching techniques on databases [23] and on XML documents [17] typically rely on the construction of specialized indexes on the instances. These indexes are used to identify at run-time the database objects corresponding to the keywords.

The main contribution of this paper is a modeling language using XML and XMI technologies aiming the implementation of a Design Patterns Management System (DPMS) for the categorization, the management and the interrogation of an extensible knowledge base, initially concerned with the 23 Design Patterns of the GoF but extensible to cover other patterns and eventually other formalisms. We implemented a support tool as a Wizard to help searching and selecting Design Patterns classified by their applicability part. Indeed, this part is devoted to what are the situations in which the Design Pattern can be applied, and how can you recognize these situations [12]. Thus, such tool remains important especially with the amount of Design Patterns actually existing.

This paper is organized as follows: Section 2 is an introduction to Design Patterns. In the main section of this paper (Section 3) we detail our modeling approach to Design Patterns. Section 4 is devoted to the Design Pattern management system (DPMS). Section 5 presents our auxiliary support tool presented as a wizard. We end the paper with a conclusion and perspectives (Section 6).

2. A SUMMURY ON DESIGN PATTERNS

Design Patterns are a mechanism for expressing object-oriented design experience [13]. A pattern for software architecture describes a particular recurring design problem that arises in specific design contexts and presents a well-proven generic scheme for its solution. The solution scheme is specified by describing its constituent components, their responsibilities and relationships, and the ways in which they collaborate [8]. They play many roles in the object-oriented development process: they are a way of reusing abstract knowledge about a problem and its solution. They give the description of the problem and the essence of its solution.

Part of the idea of Design Patterns is that patterns have a certain literal form. In the GoF book, patterns typically have these major elements: Intent, Motivation, and Applicability which concerns our purpose in this paper, Structure, Participants, Collaborations, Consequences, Implementation, Sample Code, Known Uses and Related Patterns. Part of the benefit of patterns rises from the discipline of having to verbalize these various aspects. We suppose that textual description like intent or applicability can help understanding the purpose of the Design Pattern.

2.1 Composite and Decorator patterns

The Composite allows the implementation of a scheme of tree structure. Any structure requiring a nesting of content can call this Design Pattern.

As example of a Java implementation, there are two classes Component and Container. This scheme operates in a circular fashion the strength of two possible connectors: inheritance and aggregation:

The **add** of the Container enable introducing in the Composite (Container) different component (the aggregation).

The Composite (extends Component)
⇓
We can put a Composite into another Composite
⇓
Tree structure.

In the Schema of the GoF, the methods add(), remove() and get() (size() should be also added) are defined empty in the Component and properly redefined in the Composite (but not redefined in the class Leaf). In Java, these methods are rather defined in the Composite class (Container). An example is a *display operation* (*explorer*): to explore a tree just view node information and explore (recursively) the child nodes. The Decorator has a different objective: the ability to overlay Decors applied to a Component. This implies as a ***rule*** that: ***The Decorator component is itself a component***. So, (using inheritance and Aggregation) if you take a Component C_1 and a Decor D_1, D_1 applied to $C_1 \Rightarrow D_1(C_1)$ a new component accepting to be decorated itself. If we take a new decor D_2, then we can do: $D_2(C_1)$, but more importantly: $D_2(D_1(C_1))$.

2.2 Consequences

The Pattern *Composite* is applicable whenever you want to represent part-whole hierarchies of objects and ignore the difference between compositions and individual objects. The Pattern *Decorator* is applicable whenever you want to add responsibilities to individual objects dynamically without affecting other objects.

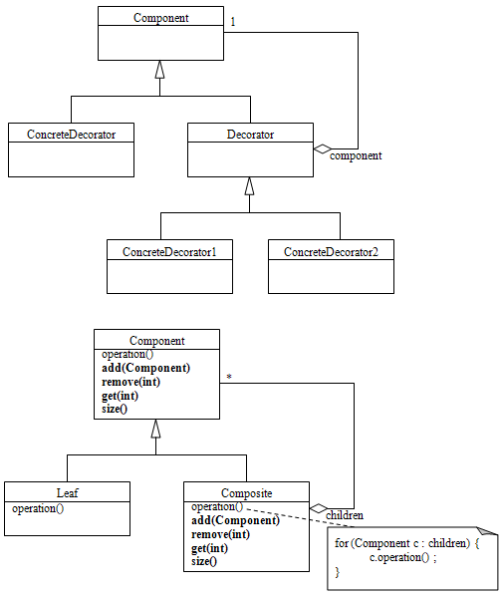

FIGURE 1: The structure of the patterns composite and decorator.

We note that the structure of the pattern composite resemble to the pattern decorator (Fig.1). However, their situations when to use (Applicability) are different. Another example, which illustrates this situation, can be raised for the Adapter and the Proxy patterns (See Fig.2 for the structure of this patterns and Tab.1 for the corresponding Applicability description).

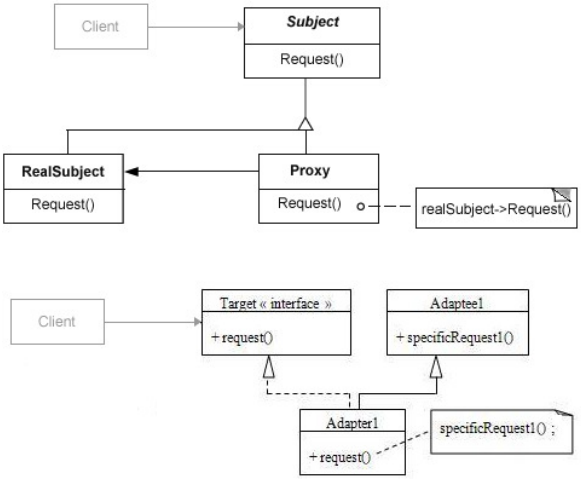

FIGURE 2: The structure of the patterns Adapter and Proxy.

The table Tab.1 expose a summary of the textual description of some Design Pattern as mentioned in the GoF book (mainly Applicability description).

Name	Intent	Applicability (When to use)
Composite	* Compose objects into tree structures to represent part-whole hierarchies. *Treat individual objects and compositions of objects uniformly.	* you want to represent part-whole hierarchies of objects. * you want clients to be able to ignore the difference between compositions of objects and individual objects. Clients will treat all objects in the composite structure uniformly.
Decorator	* Attach additional responsibilities to an object dynamically. * Provide a flexible alternative to subclassing for extending functionality.	* to add responsibilities to individual objects dynamically and transparently, that is, without affecting other objects. * for responsibilities that can be withdrawn. * when extension by subclassing is impractical.
Adapter	* Convert the interface of a class into another interface clients expect.	* you want to use an existing class, and its interface does not match the one you need. * you want to create a reusable class that cooperates with unrelated or unforeseen classes, that is, classes that don't necessarily have compatible interfaces. * (object adapter only) you need to use several existing subclasses, but it's impractical to adapt their interface by subclassing every one. An object adapter can adapt the interface of its parent class.
Proxy	* Provide a surrogate or placeholder for another object to control access to it	* whenever there is a need for a more versatile or sophisticated reference to an object than a simple pointer. * create expensive objects on demand (virtual proxy); * controls access to the original object (Protection proxy); * replacement for a bare pointer that performs additional actions when an object is accessed (Smart reference); * Persistence, counting references, checking real object.

TABLE 1: A summary table of the description of Design Patterns.

3. A MODELING LANGUAGE FOR DESIGN PATTERNS

Design Patterns community is constantly growing. From author to author, the representation of the patterns takes different forms so that documentation of new ones is becoming a problem, making it difficult for readers and reviewers to understand the patterns. For this reason we have opted for the use of new technologies to give a declaration for patterns: we use XML and XMI technology to model Design Patterns.

XML remains a suitable structured language for modeling Design Patterns: several possibilities are available to query the knowledge base through tools like XQuery, XPointer and XLink. For the object-oriented side of Design Patterns we take advantage from the power of XMI to model a part of the patterns so that we can be able to interchange knowledge and objects, generate code or produce UML diagrams [20]. The use of UML greatly facilitates the development of application in object-oriented languages to help establish a formal model and easy to understand them.

XML modeling of a Design Pattern

When modeling Patterns with XML, we have ensured that the structure is given in such a way that patterns may be easily and effectively understood by others.

We propose that patterns are represented by a single XML file divided into two main sections: The first covers the major elements of description of Design Patterns related in the GoF book and a set of metadata which characterize the pattern. The second deals with the object-oriented side of the pattern.

Pattern: the root element

The pattern is contained within the tags <*pattern*> and </*pattern*>. It consist of the two required sections cited above.The ***pattern*** element may contain other information like the formalism (GoF in our case, P.Coad, Portland, P-Sigma...) and the category (Behavioural, Creational, Structural). An example of this for the Abstract Factory pattern:

```
<pattern name="Abstract Factory" formal="GoF"
  category="Creational Pattern" id="abstractfactory">
  ...
</pattern>
```

The first part, represented by the <*heading*> element, covers the textual description of the pattern and must contain a set of required components which occur in this order: Intent, Applicability, Consequences, Problem, Solution, aliases, Related Patterns and References.

Intent

Intent element describes the intent of the pattern and looks for the *Decorator* pattern like:

```
<intent>
      <li>Attach additional responsibilities to an object dynamically</li>
      <li>Provide a flexible alternative to subclassing for extending
functionality</li>
</intent>
```

The same occurs for the ***Consequences***, the ***Problem*** and the ***Solution*** elements.

Applicability

Applicability element allows us, through a set of criteria we have developed for this reason, the query of the knowledge base not only for documentation purposes but also for assistance as explained in section 5. A fragment of this section for the *Chain Of Responsibility* pattern looks like:

```
<applicability>
    <critera>
        <statement>more than one object may handle a request,
                   and the handler isn't  known a priori.
                   The handler should be ascertained</statement>

        <keyword>handle</keyword>
            ...
    </critera>

    <critera>
     ...
    </critera>
</applicability>
```

Aliases

Aliases element list the other names that the pattern might be known (know as). An example for the *Command* pattern:

```
<aliases>
    <alia>Action</alia>
    <alia>Transaction</alia>
</aliases>
```

Related Patterns

Composed of ***RelatedPatterns*** element, it identifies the list of all the closely related patterns as shown for the *Proxy* pattern in the next example:

```
<RelatedPatterns>
    <dpattern key="adapter"/>
    <dpattern key="decorator"/>
</RelatedPatterns>
```

References

Represented with ***localReferencesm*** it's an optional metadata element dealing with a set of metadatas for the purpose of documentation, bibliography and administration based on the Dublin core formalism [10] like URL, Author information, Date, Source, Language, etc.

XMI.content

The second part presents the object-oriented modeling of the Design Pattern. This section is crucial; it presents the structure of the Design Pattern, the classes and/or objects participating in the Design Pattern and their responsibilities. For this reason, we use ArgoUml [4]: a software for creating UML diagrams. We have chosen ArgoUml because it's a language to describe objects and their relationships, a free license and programmed in Java. It allows the creation of UML diagrams in a simple and graphic way, the export of these diagrams in various formats

(XMI in our case), the generation of Java classes and the analysis of existing ones[1]. This allows us -in addition to the abilities offered by XML for query and management- it allows code generation, representation of UML diagrams and inverse engineering.

Here is an example of the representation of the **XMI.content** element for the *Singleton* pattern using ArgoUml:

```
<UML:Class xmi.id = ''
  name = 'Singleton' visibility = 'public'
  isSpecification = 'false' isRoot = 'false'
  isLeaf = 'false' isAbstract = 'false' isActive =
  'false'>
  <UML:Classifier.feature>
    <UML:Operation xmi.id = ''
      name = 'Singleton' visibility = 'private'
      isSpecification = 'false' ownerScope = 'instance'
      isQuery = 'false' concurrency = 'sequential'
      isRoot = 'false' isLeaf = 'false'
      isAbstract = 'false'>
      <UML:BehavioralFeature.parameter>
        <UML:Parameter xmi.id = ''
          name = 'return' isSpecification = 'false'
          kind = 'return'/>
      </UML:BehavioralFeature.parameter>
    </UML:Operation>
    <UML:Attribute xmi.id = ''
      name = 'instance' visibility = 'private'
      isSpecification = 'false' ownerScope = 'instance'
      changeability = 'changeable' targetScope =
      'instance'>
      <UML:StructuralFeature.multiplicity>
        <UML:Multiplicity xmi.id = ''>
          <UML:Multiplicity.range>
            <UML:MultiplicityRange xmi.id = ''
              lower = '1' upper = '1'/>
          </UML:Multiplicity.range>
        </UML:Multiplicity>
      </UML:StructuralFeature.multiplicity>
      <UML:StructuralFeature.type>
```

[1] The use of argoUml isn't necessary at itself but helps considerably to achieve this part.

4. THE DESIGN PATTERN MANAGEMENT SYSTEM (DPMS)

The solution we propose consists in the implementation of a Design Patterns management system modeled in XML format with respect to the XMI specification [21], [22]. This environment consists of three main layers Fig.4:

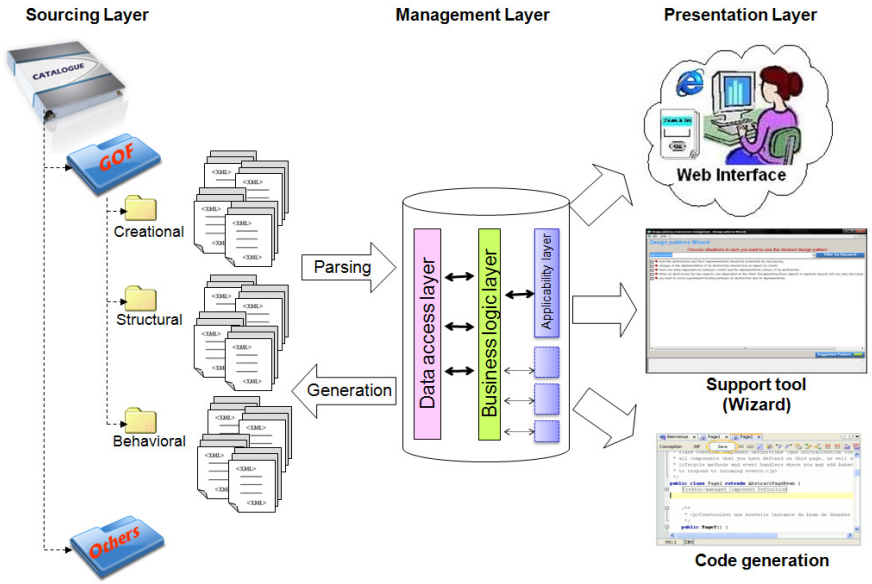

FIGURE 4: Design Pattern Management System.

4.1 Sourcing Layer

This is a storage layer for our Design Pattern knowledge base that contains all possible catalogues (GoF in particular) divided into groups of patterns according to their types (creational, structural, and behavioural). Several options were available for us to handle the base. We chose to represent each Design Pattern by a single XML file provided with identifiers that characterize the formalism and the category to which it belongs. We provide each Design Pattern with a set of metadata for bibliographic and documentation purposes. We intend to put the base in a web environment so advanced users can remotely manage the base (add new Design Patterns or eventually new formalism) and it will be wise to recognize their efforts. We implemented a framework for automatic storage and retrieval of patterns according to the modeling language discussed above. Fig.5 shows a local GUI interface for the management of Design Patterns.

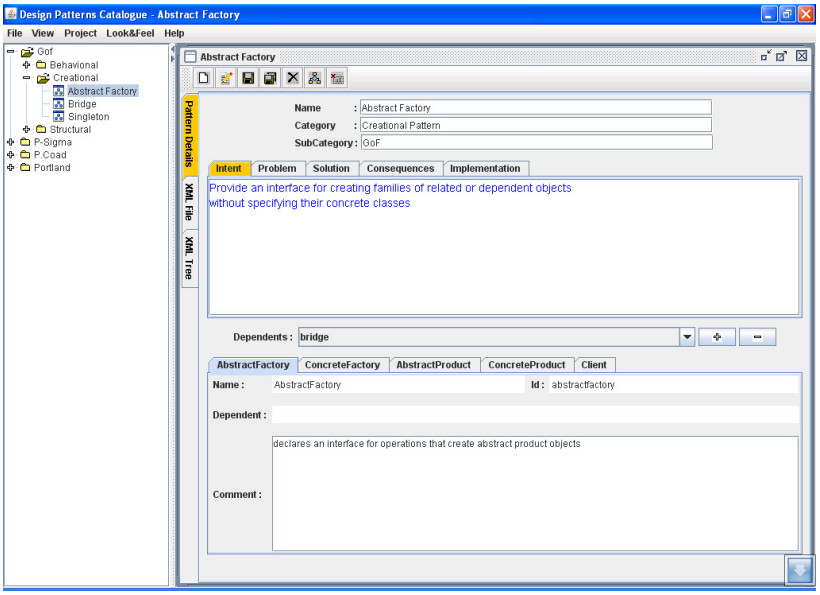

FIGURE 4: A GUI interface for storage and retrieval of patterns.

4.2 Management Layer

Extracting data from (parsing) or to the knowledge base (generation) is done through XML parsers. The management layer consists of two sub-layers business and persistence responsible for routing all requests users implementing traffic <<*request, response*>> between the base and users.

In the business layer we implemented a sub-layer (Applicability Layer) which filters knowledge according to Applicability criteria. The queries are a bit complicated since it treats a literal domain involving textual searching in data based on keywords [5].

There is a hidden side in this layer: the keywords database dedicated to parameterize the *Applicability description* of Design Patterns is managed in this level Fig.6. In fact it consists of an XML database dedicated to store all the related keywords. We opt for simplicity in choosing this method. Another solution consist of the use of ontology like this used in [15]. But, our approach relies on a common representation and storage in XML format of pattern description and the availability of a keyword-based search engine to query the pattern repository and this retrieval technique seem to be sufficient.

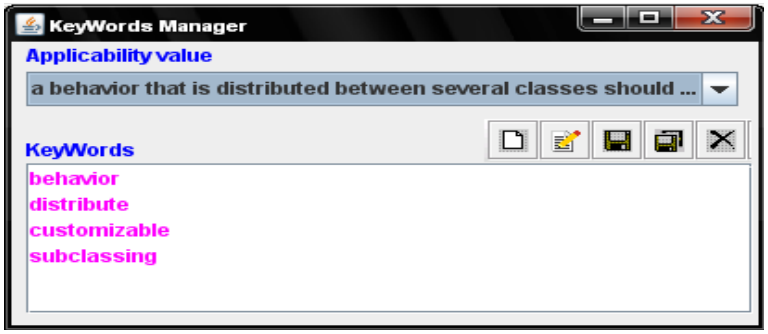

FIGURE 6: Screenshot of the keywords manager interface.

4.3 Presentation Layer

The presentation layer allows querying the knowledge base not only for bibliographic purposes but also for editing and updating catalogues, interacting with the base locally through a Java platform for administrators and a web environment for end users. It, mainly, provide support assistance to developers and other functionalities dealing with object-oriented side of the framework like code generation, UML representation or object interchange with third applications.

The user interface may be a standalone and/and a web solution: Fig.7 shows a web interface for the management of the Design Patterns knowledge base. Readers can browse the list of formalisms and the corresponding patterns. Advanced users may add new ones online. Here, we believe that modeling pattern in XML should be standardized to meet the needs of patterns authors and readers. Our attempt may be a first step in the modeling of Design Patterns in a structured language as XML.

Locally, this layer enables administrators to manage patterns through the Framework we developed to this end. Several features are available to handle the object-oriented theory and to produce components that interact with database system: The use of XMI language allows exchanging UML models from one tool to another, we take this opportunity to generate class diagram and thereby generate the source code.

FIGURE 7: Web interface for DPMS.

5. THE DESIGN PATTERN WIZARD

As mentioned in the early section, we present a support tool based on the applicability part of the description of Design Patterns. We collect a set of criteria from the description of Design Patterns defined in the GoF book classified by their applicability. In order to extract knowledge from this base we opted for a categorization into a database of pertinent keywords which characterize the statement criteria. This is mainly restricted to the applicability part of Design Patterns and can easily be extended to cover other literal description parts. The set of

criteria and the corresponding keywords database must be thinking out enough and enough to cover the most knowledge of Design Patterns. Nevertheless, the structure part of Design Patterns doesn't interest our purpose in this stage.

To exploit the platform that we built, the tool provides the end-user with a non-exhaustive list of keywords, that we have collected, to be used automatically or suggests others and the tool will help him to choose the appropriate Design Patterns.

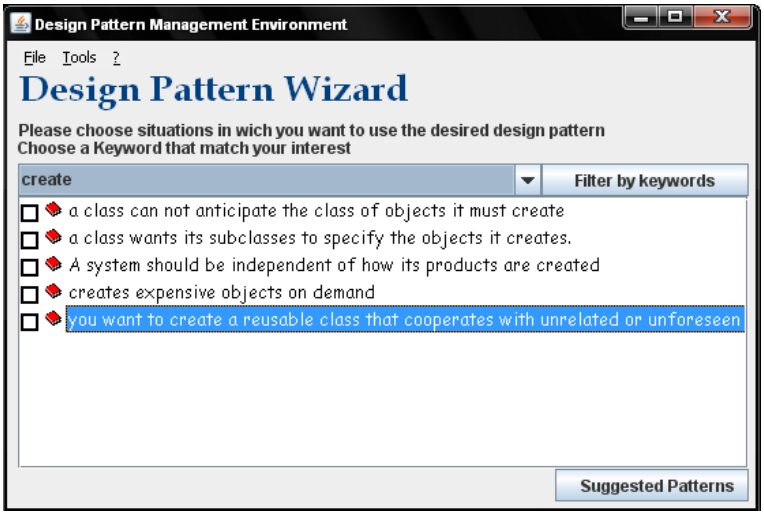

FIGURE 8: Design Pattern Wizard interface.

The wizard is a java platform which enables the search and the selection of suitable Design Patterns regards to the situations in which to use the desired Design Pattern.

The first constraint involves the selection of keywords that match the scope of the user interest Fig.8. It's an important phase in which the operation intends the filtering and the refining of the user's ideas in order to reduce the search field and have closely results to the desired ones.

With the second constraint, the program will suggest a list of all the situations matching the selected keyword. The user is required to read these criteria and select those that best describe the situations he query for. By checking appropriate statements the user is ready to generate the suitable Design Patterns. Therefore, the user may use all the functionalities available like generate code or class diagrams as shown in Fig.9.

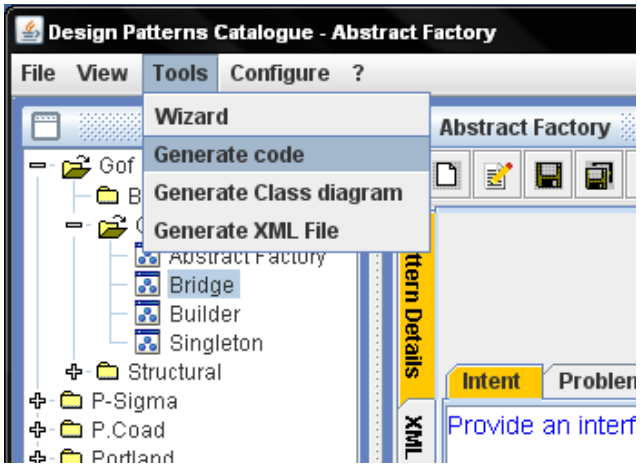

FIGURE 9: Some functionalities of DPMS framework.

6. CONCLUSION AND PERSPECTIVES

In this paper, we have presented a first attempt to initiate a modeling language of design pattern using XML and XMI technologies and a keyword-based database providing terminology for formulating applicability description of design patterns. We have implemented a prototype system, the Design Pattern Management System: an environment for computer-aided design pattern selection. The approach relies on a common representation and storage in XML format of pattern applicability descriptions and the availability of a keyword-based search engine to query the pattern repository. With patterns being represented using structured information, Design Patterns become, literally, easy to understand and to be documented and formally able to interchange knowledge and objects. The keyword-based database constitutes the basis of the Design Pattern Wizard, an automated tool to support software developers to search and chose the Design Pattern adequate to their design problems.

The work presented in this paper enables software developers to find the adequate design pattern(s) for a given design problem. It remains, thus, a measure avoiding the impact of the vast amount of design patterns present in different design pattern catalogues. The keyword-based database can be improved to cover other design patterns beyond the GoF book. In order to enlarge the field of keywords we put forward an ontology based on the applicability description of design patterns which, with the test of the Wizard with other catalogues of design patterns remains for future work.

7. REFERENCES

[1] Admodisastro, N. S. Palaniappan (2002). A code generator tool for the gamma Design Patterns. Malaysian Journal of Computer Science 15 (2), 94-101.

[2] Albin-Amiot, H. (2003). Idiomes et patterns Java: Application à la synthèse de code et la détection. Ph. D. thesis, _Ecole des Mines de Nantes and Université de Nantes.

[3] Alexander, C., S. Ishikawa, M. Silverstein (1977). A Pattern Language, volume 2. Center for Environmental Structure Series: Oxford University Press, New York, NY.

[4] ArgoUml (2010). ArgoUml Modeling Tool, http://argouml.tigris.org/.

[5] Bergamaschi, S., E. Domnori, F. Guerra (2010). Keymantic: Semantic keywordbased searching in data integration systems. Proceedings of the VLDB Endowment 3 (2).

[6] Borne, I. N. Revault (1999). Comparaison d'outils de mise en œuvre de Design Patterns. Object-oriented Patterns, Vol5, Num2.

[7] Budinsky, F., M. Finnie, J. Vlissides, P. Yu (1996). Automatic code generation from Design Patterns. IBM Systems Journal, IBM Corp 35 (2), 151-171.

[8] Buschmann, F., R. Meunier, H. Rohnet, P. Sommerlad, M. Stal (1996). Pattern-Oriented Software Architecture, Volume 1, A System of Patterns. New York: Jhon Wiley & Sons, Ltd.

[9] Conte, A., M. Fredj, I. Hassine, J. Giraudin, D. Rieu (2002). Agap : an environment to design and apply patterns. In IEEE International Conference on Systems, Man and Cybernetics (IEEE SMC), Hammamet, Tunisia.

[10] DC. Dublin Core Speci_cation, http://dublincore.org/speci_cations/.

[11] EDEN, A. (2001). Formal speci_cation of object-oriented design. In International Conference on Multidisciplinary Design in Engineering, CSME-MDE 2001, November 21-22, Montreal, Canada.

[12] Gamma, E., R. Helm, R. Johnson, J. Vlissides (1995). Design Patterns: Elements of Reusable Object-Oriented Software. New York: Addison-Wesley Professional Computing series, Addison-Wesley Publishing Company.

[13] Gamma, E., R. Helm, R. Johnson, J. Vlissides (2002). Design Patterns: abstraction and reuse of object-oriented design. Software pioneers, Broy, M., Denert, E. (Eds.) LNCS. 1, 701-717 Springer{Verlag New York, Inc., New York, NY, USA.

[14] Hannemann, J. G. Kiczales (2002). Design Pattern implementation in java and aspectj. Proceedings of the 17th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications, OOP-SLA '02 37, 161-173, New York, NY, USA.

[15] Kamp_meyer, H. S. Zschaler (2006). Finding the pattern you need: The Design Pattern intent ontology. Engells, G. et al. (eds.) MoDels 2007, LNCS. Springer, Heidelberg 4735, 211-225.

[16] Kim, S.-K. C. David (2009). A formalism to describe Design Patterns based on role concepts. Formal Aspects of Computing 21 (5), 397{420.

[17] Liu, Z., J.Walker, Y. Chen (2007). Xseek: A semantic xml search engine using keywords. VLDB, ACM, 1330-1333.

[18] Taibi, T. D. Check Ling Ngo (July-August (2003)). Formal specification of Design Patterns - a balanced approach. Journal of Object Technology 2 (4), 127-140.

[19] Tichy, W. F. (1997). A catalogue of general-purpose software Design Patterns. TOOLS '97, Proceedings of the Tools-23: Technology of Object-Oriented Languages and Systems, IEEE Computer Society, Washington, DC, USA.

[20] UML (2008). Unified Modeling Language, http://www.omg.org/spec/uml/.

[21] XMI (2007). XML Metadata Interchange : OMG formal 2007-12-01, http://www.omg.org/spec/xmi/.

[22] XML (2008). eXtensible Markup Language 1.0 (_fth edition) http://www.w3.org/tr/2008/rec-xml-20081126.

[23] Yu, J., L. Qin, L. Chang (2009). Keyword search in databases. In Synthesis Lectures on Data Management, Morgan & Claypool Publishers.

[24] Zimmer, W. (1994). Relationships between Design Patterns. In Pattern Languages of Program Design. Addison-Wesley.