A Framework for Evaluating Software Design Pattern Specification Languages

Salman Khwaja and Mohammad Alshayeb
Information and Computer Science Department
King Fahd University of Petroleum & Minerals
Dhahran 31261, Saudi Arabia
{khwaja, alshayeb}@kfupm.edu.sa

Abstract—**Software design patterns are solutions to recurring problems. These solutions have been widely accepted and adopted by the software practitioners. Design Patterns were initially described informally through the use of textual descriptions or some object oriented graphical notations. However, informal descriptions of design patterns give rise to ambiguity and incorrect usage of the design pattern. Many design pattern specification languages have been proposed to address the shortcomings of informal descriptions. These design pattern formalization techniques have been developed for different intentions including verification of design patterns and detection of design patterns in source code. In this paper we provide a set of evaluation criteria to evaluate and compare the various design pattern specification languages in order to aid practitioners and researchers to select the appropriate language for their use.**

Keywords—design patterns; design pattern specification languages; comparison framework

I. INTRODUCTION

The complexity of a software problem can be managed by breaking down of the problem into smaller sub-problems. Complex systems are built by using smaller parts. Design patterns provide knowledge in an accessible way to provide reusable solutions to these sub-problems [1].

The purpose for using design patterns in software development is to improve the reusability and the quality of software. The motivation for formalizing design patterns is to make them easier to understand and implement in software applications.

As the use of the software design patterns is on the rise [2], many design patterns specification languages have been proposed to provide support for a consistent, unambiguous and a simple way of sharing design patterns knowledge. A design pattern language should have the flexibility of defining the design pattern in a very generic term by the software designers, and it should cover different instances of that design pattern. Also, the design pattern language should provide the capability to define the design pattern in such a way that can help software developers to create an exact replica of the design pattern during implementation.

Several techniques and approaches have been developed for the specification of the design pattern languages. Many design pattern specification languages (DPSLs) have been proposed in the literature. These DPSLs are proposed with different intention with varying features and capabilities and covering many or all aspects of the design patterns structures and behaviors. However, so far there is no comprehensive framework developed for evaluating these DPSLs. Therefore, in this paper we propose a framework to help in evaluating and comparing various DPSLs in order to aid practitioners and researchers to select the appropriate language for their use.

II. RELATED WORK

Cunningham and Beck used Alexander's [3] ideas to develop a small pattern language for guiding Smalltalk programmers [4]. Coplien compiled a catalog of C++ idioms (which are one kind of pattern) and later published them as a book [5]. Shortly thereafter, the Gang of Four's Design Patterns book [6] was published.

Several design pattern languages have been proposed; however, each has distinct features. Some of these languages are based on formal mathematical techniques [7, 8]. Other languages are based on UML notation [9, 10] or ontology [11, 12]. However, no framework was developed to evaluate these different design pattern specification languages.

III. DESIGN PATTERN SPECIFICATION LANGUAGES CATEGORIZATION

DPSLs can be categorized based on the primary motivation for their development. This motivation or intent determines many of their features. Although the list of the intent for the DPSLs can be long, but they can be broadly categorized in the below four categories. One design pattern specification language can fall into multiple categories too, as sometime there are multiple motivations or purposes for the DPSL. These four categories are:

A. Defining and Describing Design Patterns

The earliest DPSLs were created for defining and describing design pattern in a formal or semi-formal way. One of the first examples belonging to this category is LePUS [13]. LePUS is based on textual higher order monadic logic to express solutions. Mak et al. [14] proposed an extension to LePUS which solves the issue of pattern composition in LePUS. Many of the DPSLs based on the mathematical formalism belong to this category of the design pattern.

Some of the modeling based DPSLs fall in this category also. Examples include DPDL [15]. DPDL tries to cover

978-1-4799-0174-6/13/$31.00 ©2013 IEEE

characteristic of design pattern using XML. Constraint Diagrams use visual modeling to describe design patterns achieving greater precision without utilizing obtuse mathematical symbols [16]. Balanced Pattern Specification Language (BPSL) [17] was designed to cope with the shortcomings of the existing formal approaches for pattern specification. BPSL formally specify the structural as well as behavioral aspects of patterns at three levels of abstraction: pattern composition, patterns, and pattern instances.

B. Detection of Design Patterns

One of the major efforts in software design patterns lately is on the design pattern detection in the source code. Many of the DPSLs are created for detecting design pattern in source code. SPINE [18] is loosely based on Prolog. Patterns are detected through HedgeHOg proof engine using the definition provided in the SPINE.

Most of these languages are restricted to detect design pattern from a specific programming language. Costagliola et al. [19] presented a visual language based pattern recovery approach for extracting design patterns from C++ examples.

C. Verification and Validation of Design Patterns

A new endeavors in the DPSLs is on the verification and validation of the design patterns. Saeki [20] used LOTOS for specifying patterns that appeared in Gamma et al. [6] and their composition.

Some languages which are linked with repositories have more emphasis on the verification and validation of the design patterns. RSL [21], based on the RAISE specification languages, shows how design can be formally linked with the patterns, how properties of individual patterns can be specified and provides basis for formally checking whether design and patterns are consistent with each other.

D. Graphical Modeling of Design Patterns

Still other design pattern languages try to represent design pattern in graphical form. This graphical form could be any of the UML diagrams, extensions to UML diagrams or new graphical representation of the design pattern. Examples include Role Based Meta-modeling Language (RBML) [9] proposed by Kim and Dae. It is a meta-modeling based technique for specification of structure, interactions, and state-based behavior of a design pattern. Design Pattern Modeling Language (DPML) is a domain-specific visual language that offers a higher level of abstraction and representation, for design-level constructs [10].

E. Other Design Pattern Specification Languages Categorization

DPSLs can also be categorized based on the basis of design pattern specification languages and based on the notation of design pattern specification languages as described below.

- **Categorization Based on Basis of Design Pattern Specification Languages:** A categorization of DPSLs can be done based on the basis of the design pattern specification language. In this categorization, DPSLs can be categorized into three types [15]: languages based on mathematical formalism, languages based on modeling languages and languages based on other languages.
- **Categorization Based on Notation of Design Pattern Specification Languages:** Another categorization of DPSLs is based on the notation of the design pattern specification language. The notation of the design pattern language determines the features of the design pattern languages. A graphical notation based only, design pattern language cannot be too formal, so they are mostly semi-formal or informal design pattern languages. In this categorization we also have three types: textual based design pattern languages, graphical notation based design pattern languages and amphibious design pattern languages.

IV. Design Pattern Specification Languages Evaluation Framework

In this section, we propose a framework to evaluate the DPSLs. The proposed framework consists of two parts. The first part deals with the general aspects of the DPSLs. The properties in this section evaluate the general aspects of the DPSLs and are not concerned with the capabilities of the DPSLs. We call this section the general properties. The Second section contains the core properties of the languages. In the below subsections we describe the properties of each part.

A. General properties

- **The basis:** The basis of the design pattern languages can be classified into four categories: 1) languages that are based on formal mathematical logic, these languages use mathematical formalism for design patterns, 2) languages based on UML, 3) languages based on general purpose programming languages, and 4) languages based on ontology that are based on semantic web technologies.
- **Integratable in IDEs:** Can the language be integrated with an Integrated Development Environment (IDE). In general, formal languages have less chance of having this feature.
- **Platform independence:** Is the DPSL platform independent i.e. can it work on different platform or is it platform specific [19]. The languages which are targeted towards describing and defining design pattern are usually platform independent.
- **Template support:** Template support means that a design pattern language should provide templates which can then be used to create a specific instance of that design pattern. This property is dependent on the objective of the design pattern language. Design pattern languages which are designed for the

identification of design patterns in the source code, usually lack this property.

- **UML support:** Does the DPSL support UML? UML is the most common and widely used graphical output to represent software architecture and design in a graphical view. Therefore, most of the design pattern languages use UML to show the design pattern in a graphical view. Some design pattern languages have developed their own graphical notation to depict a design pattern in a graphical view.
- **Graphical support:** Does the design pattern language provide any graphical output in any format other than UML?
- **Learning curve for programmers:** Design patterns are ultimately used by programmers to provide a tested solution to a commonly occurring problem. But some design pattern languages have focused on the formal/mathematical approach that makes these languages incomprehensible by the programmers. Learning curve is based on the amount of time needed to not only learn how to make design pattern in the language but also the amount of time needed to learn the basics of the design pattern language to make that design pattern. So the languages with which the programmers are accustomed are given low learning curve value and others are given medium or high.
- **Target:** This property describes the intended target use of the design pattern language.

B. Core Properties

This section describes the properties of the design pattern specification languages that express the design pattern in a formal or informal way. Properties in this core section belong either to conciseness, comprehensiveness, formalism or expressiveness.

- **Ability to address object-oriented (OO) paradigm:** The design patterns are based on the interaction of the classes in some specific way. If a design pattern language does not have a full capability of capturing the relationship among classes then it will not be able to describe design patterns adequately. Therefore, all the design pattern languages should fulfill this criterion.
- **Ability to handle multiple entities:** This criterion checks if the design pattern language is able to handle multiple entities (classes, objects, variables etc.) of the same type by grouping them together.
- **Capability to identify collaborations distinctly:** Design patterns are the collaboration of different parts of a system in a specific way. Although all design pattern languages handle collaboration inside a design pattern but this property checks if the collaboration can be identified distinctly in the language
- **Capability to identify participants distinctly:** This feature identifies the design patterns participants which are required in a particular design pattern. The participants in a design pattern are used for depicting the structural and the behavioral aspects of the design pattern.
- **Capability to identify the structure distinctly:** Formalizing the design pattern according to the structure is helpful if the design pattern is required to be shown by class diagram. Therefore this feature is important if the design pattern language is required to create UML diagrams.
- **Capability to formalize OO patterns:** This criterion cheeks if the design pattern language is expressive enough to handle design patterns in OO domain including the one mentioned in the "Gang of Four" design patterns. Some design pattern languages are very specific and are created for specific types of design patterns.
- **Textual notation provided by the language:** This property checks if the design pattern language is based on visual aids/diagrams/notations or it has a formalized textual syntax to represent the design pattern. Some design pattern languages are based on UML or some other visual notations and have no formal textual syntax.
- **Graphical notation provided by the language:** This property checks the inverse of the previous property, i.e. it checks if the design pattern language has a graphic notation. Some textual design pattern languages do not have a graphical notation and they do not produce graphical output.
- **Original formalism:** Some design pattern languages are extensions of previous effort. These design pattern languages improve or extend or remove some short comings of the previous design pattern language. Such design pattern languages are identified by this property.
- **Supported views:** This property checks which views does the design pattern language is capable to handle: structural, behavioral and functional views. Some design pattern languages can only support one or two of these views.
- **Other features:** It checks whether the design pattern language is able to formalize the design pattern in aspect other than the one mentioned above. For example the LePUS design pattern language is able to formalize the design pattern in a form of formula.

V. CONCLUSION

With the increase of software complexity and requirement for swift delivery of software, the use of the software design pattern increases. This will increase the use of the software DPSLs. Therefore, there is a need for a framework to be able to analyze, compare and evaluate the

DPSLs. This framework can help the software architects to select the DPSL which best fulfills their requirements. In this paper we proposed a framework to evaluate the available DPSLs in the literature. The framework consists of two groups; the basic group is concerned with the general aspect of the DPSLs while the other group is concerned with the core properties of the languages. Figure 1 shows the proposed framework. The figure shows that the basic properties group includes ease of use and support, and the core properties group includes: conciseness, comprehensiveness, formalism and expressiveness.

Future work includes conducting a detailed survey of the design patterns specification languages and use this framework to evaluate these languages.

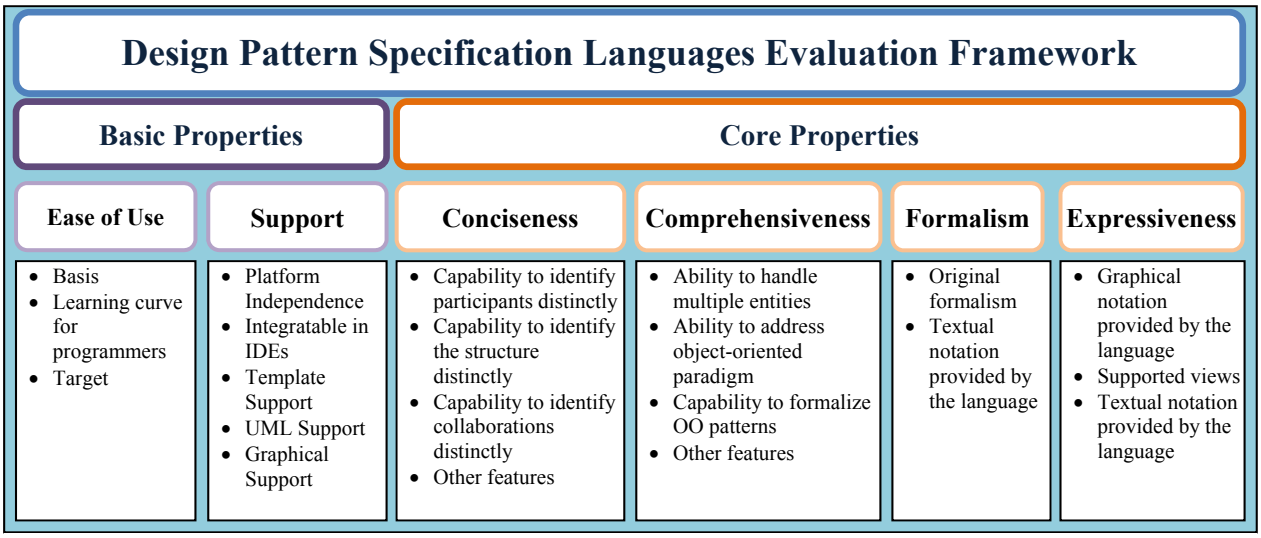

Figure 1. The proposed Design Pattern Specification Languages Evaluation Framework

ACKNOWLEDGMENT

The authors acknowledge the support of the Deanship of Scientific Research of the King Fahd University of Petroleum and Minerals for the development of this work.

REFERENCES

[1] M. C. Payne, "Enhancing Design Pattern Languages for Automatic Software Design Pattern Detection," 2006.

[2] J. Dong, P. Alencar, and D. Cowan, "Formal Specification and Verification of Design Patterns," in *Design Pattern Formalization Techniques*, ed: IGI Global, 2007, pp. 94-108.

[3] C. Alexander, *The Timeless Way of Building*: Oxford University Press, 1979.

[4] K. Beck and W. Cunningham, "Using Pattern Languages for Object Oriented Programs," OOPSLA - Conference on Object-Oriented Programming, Systems, Languages, and Applications1987.

[5] J. O. Coplien, *Advanced C++ Programming Styles and Idioms*: Addison-Wesley Pub (Sd), 2002.

[6] R. H. Erich Gamma, Ralph Johnson, John Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*: Addison Wesley, 1994.

[7] E. Gasparis, "LePUS: A Formal Language for Modeling Design Patterns," in *Design Pattern Formalization Techniques*, ed: IGI Global, 2007, pp. 357-372.

[8] R. R. Raje, S. Chinnasamy, A. M. Olson, and W. Hidgon, "The Applications and Enhancement of LePUS for Specifying Design Patterns," in *Design Pattern Formalization Techniques*, ed: IGI Global, 2007, pp. 236-257.

[9] D.-K. Kim, "Role-Based Metamodeling Language for Specifying Design Patterns," in *Design Pattern Formalization Techniques*, ed: IGI Global, 2007, pp. 183-205.

[10] D. Mapelsden, J. Hosking, and J. Grundy, "Design pattern modelling and instantiation using DPML," in *CRPIT '02: Proceedings of the Fortieth International Conference on Tools Pacific*, ed. Sydney, Australia: Australian Computer Society, Inc., 2002, pp. 3-11.

[11] A. Gangemi, "Ontology Design Patterns for Semantic Web Content," in *Proceedings of the Fourth International Semantic Web Conference*, ed: Springer, 2005, pp. 262-276.

[12] J. Dietrich and C. Elgar, "An Ontology Based Representation of Software Design Patterns," in *Design Pattern Formalization Techniques*, ed: IGI Global, 2007, pp. 258-279.

[13] A. H. Eden, J. Y. Gil, and A. Yehudai, "A Formal Language for Design Patterns," Washington University, St. Louis, Missouri, USA1996.

[14] J. K. H. Mak, C. S. T. Choy, and D. P. K. Lun, "Precise specification to compound patterns with ExLePUS," 2003, pp. 440-445.

[15] S. Khwaja and M. Alshayeb, "Towards design pattern definition language," *Software: Practice and Experience,* 2011.

[16] S. Kent, "Constraint diagrams: visualizing invariants in object-oriented models," 1997, pp. 327-341.

[17] T. Taibi and D. C. Ngo, "Formal specification of design pattern combination using BPSL," *Information and Software Technology,* vol. 45, pp. 157-170, March 2003.

[18] A. Blewitt, "SPINE: Language for Pattern Verification," in *Design Pattern Formalization Techniques*, ed: IGI Global, 2007, pp. 109-122.

[19] A. De Lucia, V. Deufemia, C. Gravino, and M. Risi, "Behavioral pattern identification through visual language parsing and code instrumentation," 2009, pp. 99-108.

[20] M. Saeki, "Behavioral specification of GOF design patterns with LOTOS," in *APSEC '00: Proceedings of the Seventh Asia-Pacific Software Engineering Conference*, ed. Washington, DC, USA: IEEE Computer Society, 2000, p. 408.

[21] A. Flores, A. Cechich, and G. Aranda, "A Generic Model of Object-Oriented Patterns Specified in RSL," in *Design Pattern Formalization Techniques*, T. Taibi, Ed., ed: IGI Global, 2007, pp. 44 - 72.