DC2DP: A Dublin Core Application Profile to Design Patterns

Angélica Aparecida de Almeida Ribeiro, Jugurta Lisboa-Filho,
Lucas Francisco da Matta Vegi and Alcione de Paiva Oliveira
Departamento de Informática, Universidade Federal de Viçosa (UFV), Viçosa, Minas Gerais, Brazil

Keywords: Design Pattern, Dublin Core Standard, Reuse.

Abstract: Design patterns describe reusable solutions to existing problems in object-oriented software development. Design patterns are mostly documented in written form in books and scientific papers, which hinders processing them via computer, their diffusion, and their broader reuse. They can also be found on the internet, though documented with little detail, which makes it hard to understand and consequently reuse them. This paper presents an application profile of the Dublin Core metadata standard specific for design patterns, called DC2DP. The goal is to allow design patterns to be documented so as to provide the user with a more detailed and standardized description, besides enabling automatic processing through web services. The paper also extends an Analysis Patterns Reuse Infrastructure (APRI) by adding a design pattern repository to it, thus allowing these patterns to be cataloged and searched, which makes their discovery, study, and reuse easier.

1 INTRODUCTION

In order to locate a book in a library, a work of art in a museum, or a city map in a map repository, the various catalogs of these objects need to be consulted. In the digital era, this type of catalog corresponds to a metadata (data on data) repository containing the description and information on how to obtain or locate the documented object. Metadata must follow a standardized documentation structure (element set) so that the systems are able to achieve interoperability in its search modules. The Dublin Core Metadata Element Set (DCMI, 1998) was defined as a way to serve several areas, making available a minimum set of mandatory elements in documenting any type of object. Moreover, an application profile may be defined, i.e., a standard customization for a specific area.

In software engineering, design patterns (Gamma et al., 1995), analysis patterns (Fowler, 1997), frameworks, and components are examples of reusable computational artifacts during software development. The issue is that these reusable artifacts are not easily found and most times programmers and designers choose to develop their solutions from scratch instead of researching the existence of previously tested solutions validated in other systems.

Thus, Vegi et al. (2012b) proposed an Analysis Patterns Reuse Infrastructure (APRI) made up of a repository of analysis patterns, documented based on a Dublin Core Application Profile specific for analysis patterns.

According to Gamma et al. (1995), a design pattern identifies the mains aspects of a design structure that is common and possibly useful for creating other object-oriented projects. Each design pattern is able to focus on one particular problem or topic of an object-oriented design. The pattern describes how, where, and in which situation it must be employed, and the consequences of its use. Gamma et al. (1995) states that the object-oriented architectures, when well structured, may carry several patterns. Gamma et al. also claims that one way of measuring the quality of an object-oriented system is to assess how the developers used the common collaborations among its objects. The use of these patterns during the development of a system is able to produce smaller, simpler, and much more understandable architectures than in architectures in which these patterns are ignored.

Documenting these patterns helps to capture the design experience so that the designers can use them more effectively. For that end, these patterns must be documented and presented in some easily accessible and understandable catalog.

Aparecida de Almeida Ribeiro A., Lisboa-Filho J., Vegi L. and Oliveira A..
DC2DP: A Dublin Core Application Profile to Design Patterns.
DOI: 10.5220/0004885202090216
In *Proceedings of the 16th International Conference on Enterprise Information Systems* (ICEIS-2014), pages 209-216
ISBN: 978-989-758-028-4

This paper proposes a specific Dublin Core Application Profile to document design patterns. The profile is based on elements of the DC2AP metadata profile, proposed by Vegi et al. (2012a) to document analysis patterns, and on the template used by Gamma et al. (1995). Moreover, this paper proposes extending the APRI structure by adding to it a repository of design patterns, thus allowing this type of pattern to be cataloged and reused.

The remaining of the paper is structured as follows. Section 2 reviews works related to design pattern catalogs, besides introducing DC2AP, a metadata profile to document analysis patterns. Section 3 introduces DC2DP, a Dublin Core Application Profile to document design patterns. Section 4 proposes extending the APRI structure, while section 5 presents the final considerations and proposes some future work.

2 RELATED WORKS

2.1 Design Pattern Catalogs

A design pattern catalog is made up of a set of related patterns with characteristics in common. These patterns may be used individually or be interconnected, since they may be used alongside each other. There are several design pattern catalogs, such as GoF Patterns (Gamma et al., 1995), J2EE Patterns (Alur et al., 2003), SOA Patterns (SOA Patterns, 2013), among others.

Each existing design pattern catalog uses a way of documenting the patterns that compose it, i.e., each one uses a set of elements to describe the pattern, with no standardized way of documenting design patterns.

2.2.1 GoF Pattern Catalog

Gamma et al. (1995) propose in their book a design pattern catalog, which became known as GoF Pattern Catalog, made up of 23 patterns classified according to two criteria: scope and purpose.

Regarding scope, the patterns may be split into class and objects. As for the purpose, the patterns are classified into creation, structural, and behavioral patterns.

In order to describe the design patterns in the catalog, the authors divided each pattern into sections according to the template proposed. The elements in the template used in documenting these patterns are: pattern name and classification, purpose and goal, also known as, motivation, applicability, structure, participants, collaborations, consequences, implementation, code examples, and known uses.

2.2.2 J2EE Pattern Catalog

Alur et al. (2003) presented in their book a design pattern catalog based on the work experience with the J2EE platform of Sun Java Center to clients worldwide.

The J2EE Pattern catalog is made up of 21 patterns split into presentation layer, business layer, and integration layer patterns. Each pattern is documented following a template. The elements in the template are: problem, forces, solution, consequences, and related patterns.

2.2.3 SOA Pattern Catalog

The design patterns in the catalog presented in SOA Patterns (2013) list the service-oriented principles when a dependency or relationship among services in an architecture must be highlighted.

The SOA Pattern catalog is made up of 83 patterns divided into the following categories: Service Implementation, Service Security, Service Contract Design, Legacy Encapsulation, Service Governance, Capability Composition, Service Messaging, Composition Implementation, Service Interaction Security, Transformation, REST-inspired.

The elements in the template to document the SOA patterns are: problem, solution, application, principles, related patterns, goals related to service-oriented computing.

2.2 Dublin Core Application Profile to Analysis Patterns

The Dublin Core Application Profile to Analysis Patterns (DC2AP) was proposed by Vegi et al. (2012a) to document analysis patterns. This metadata profile was created based on the template proposed by Pantoquilho et al. (2003) to document analysis patterns and on the Dublin Core metadata standard elements (DCMI, 1998).

According to Vegi et al. (2012a), the goal of DC2AP is to improve the recovery and reuse of analysis patterns by means of a description that allows the computer to perform a more precise treatment of the data previously not recovered by search engines. This task is performed based on detailed information provided on these patterns.

The DC2AP elements are associated to a Universal Resource Identifier (URI) and

semantically described by a Resource Description Framework (RDF). Using URI and RDF enables using the Linked Data concept, which then allows the patterns to be made available in an environment where the data may be processed directly or indirectly by machines. The DC2AP elements are detailed in section 3.2.

3 DUBLIN CORE APPLICATION PROFILE TO DESIGN PATTERNS

This section describes the Dublin Core Application Profile to Design Patterns (DC2DP). In order to define the DC2DP profile elements, initially the design pattern documentation template elements proposed by Gamma et al. (1995) were compared with the elements of the DC2AP so as to analyze whether the same metadata profile could be used to document the analysis and design patterns.

Based on this comparison, a significant difference was found between the element sets since the analysis patterns are more targeted towards the phase of analysis of requirements and modeling of the problem, mainly focusing on documenting the functional and non-functional requirements, the class diagrams among other artifacts targeted towards the analysis phase of a piece of software. On the other hand, design patterns are targeted towards the solution phase of the design, in order to define object-oriented architectures, tending to document examples of source code for certain problems.

Thus, the Dublin Core Application Profile to Design Patterns (DC2DP) was specified based on the elements of the template proposed by Gamma et al. (1995) to specify design patterns and on the DC2AP elements.

DC2DP's main goal is to improve the recovery and reuse of design patterns, by providing more detailed information on these patterns, now in a recoverable by search engines format.

The next sub-sections detail the steps taken to define DC2DP. Sub-section 3.1 presents the mapping between the Dublin Core elements and those of the template proposed by Gamma et al. (1995). Sub-section 3.2 presents the mapping between the elements generated in the mapping of sub-section 3.1 and the elements of DC2AP. Finally, sub-section 3.3 presents the DC2DP elements with their application rules.

3.1 Mapping between the Elements of the Dublin Core Metadata Standard and Gamma's Template

Unlike the Dublin Core metadata standard, which is generic and, therefore, helps in documenting resources from several domains, the template proposed by Gamma et al. (1995) is specific for documenting design patterns, which is why it was chosen to be the basis for creating DC2DP.

Hence, the first task in defining DC2DP was mapping the elements of the template of Gamma et al. (1995) and the Dublin Core elements. This mapping process was carried out by comparing all the elements of either structure, based on the elements' semantic correspondence. The result of this mapping can be seen in Figure 1.

Dublin Core	Gamma et al. (1995)
Title	Name Also Known As
Creator	No Equivalent
Subject	No Equivalent
Description	Intention Motivation Applicability Known Uses
Publisher	No Equivalent
Contributor	No Equivalent
Date	No Equivalent
Type	No Equivalent
Format	No Equivalent
Identifier	No Equivalent
Source	No Equivalent
Language	No Equivalent
Relation	Related Patterns
Coverage	No Equivalent
Rights	No Equivalent
No Equivalent	Classification
	Structure
	Participants
	Collaborations
	Consequences
	Implementation
	Sample Code

Figure 1: Mapping between the elements of the Dublin Core and the template by Gamma et al. (1995).

3.2 Mapping between the DC2AP Elements and Dublin Core Profile with Gamma's Template Elements

From the mapping described in sub-section 3.1, the elements from the Dublin Core profile and from the

DC2AP	Dublin Core + Gamma et al.
1.Identifier	Identifier
2.Title 2.1 Alternative Title	Title Also Known As
3.Creator	Creator
4.Subject	Subject
5. Description 5.1 Problem 5.2 Motivation 5.2.1 Example 5.2.2 Known Uses 5.3 Context	Description Intention Motivation Applicability Known Uses No Equivalent
6. Publisher	Publisher
7. Contributor	Contributor
8. Date 8.1 Created 8.2 Modified	Date No Equivalent No Equivalent
9. Type 9.1 Notation	Type No equivalent
10. Format	Format
11. Source	Source
12. Language	Language
13. Relation 13.1 Is Version Of 13.2 Is Replaced By 13.3 Replaces 13.4 Is Part Of 13.5 Has Part 13.6 Is Designed With 13.7 Should Avoid 13.8 Complemented By 13.9 About	Relation No Equivalent No Equivalent No Equivalent No Equivalent No Equivalent No Equivalent No Equivalent Related Patterns No Equivalent
14. Coverage	Coverage
15. Rights	Rights
16. History 16.1 Event Date 16.2 Author 16.3 Reason 16.4 Changes	No Equivalent No Equivalent No Equivalent No Equivalent No Equivalent
17. Requirements 17.1 Functional Requirements 17.2 Non-functional Requirements 17.3 Dependencies and Contributions 17.3.1 Dependency Graph 17.3.2 Contribution Graph 17.4 Conflict Identification & Guidance to Resolution 17.5 Priorities Diagram 17.6 Participants	No Equivalent No Equivalent No Equivalent Colaborations No Equivalent No Equivalent Implementation No Equivalent Participants
18. Modelling 18.1 Behaviour 18.1.1 Use Case Diagram 18.1.2 Collaboration/Sequence Diagrams 18.1.3 Activity/State Diagrams 18.2 Structure 18.2.1 Class Diagram 18.2.2 Class Description 18.2.3 Relationship Descriptions 18.3 Solution Variants	Structure No Equivalent No Equivalent Structure No Equivalent Structure Structure Participants Collaborations No Equivalent
19. Resulting Context	No Equivalent
20. Design Guidelines	Implementation
21. Consequences 21.1 Positive 21.2 Negative	Consequences Consequences Consequences
No Equivalent	Classification Sample Code

Figure 2: Mapping between DC2AP and the elements of the Dublin Core + Gamma et al. (1995).

template by Gamma et al. (1995) were combined, thus originating the basic DC2DP structure.

Next, the elements of this structure were compared to the DC2AP elements, since there was a need to check whether these two patterns were very similar and whether a new Dublin Core profile had to be generated to document design patterns or just a profile based on DC2AP had to be created to document design patterns. The comparison was similar to the one made with Dublin Core, with the elements being compared according to their semantics. The result of this mapping can be seen in Figure 2.

As it can be seen, many elements of Dublin Core + Gamma to not match the DC2AP profile, thus a new standard for documenting design patterns was proposed, whose elements are described in sub-section 3.3.

3.3 DC2DP: Dublin Core Application Profile to Design Patterns

Figure 3 shows the elements belonging to the Dublin Core Application Profile to Design Patterns (DC2DP), generated based on the mappings presented in sub-sections 3.1 and 3.2.

In order to generate DC2DP, the main characteristics in the Dublin Core pattern elements were taken into account, thus allowing the design patterns to be documented based on a generic metadata standard so as to transmit to the user necessary information on the patterns, aiding them in making the right choice and appropriately using these patterns.

Gamma et al. (1995) present the elements in a catalog for design pattern documentation that the user must read in order to understand whether the pattern they use or intend to use is the right one for the problem being tackled. Some of these elements are: Applicability (Examples), Consequences, Structure (Structural Modeling), Participants (Class Description), Collaborations (Description of Relationships), Source-code Example. Therefore, these elements were kept in DC2DP.

Other elements, however, had their names changed when integrating DC2DP, which are presented in parenthesis as shown above. This change was done because the semantic meanings of the elements were similar to some elements in DC2AP, thus keeping both elements was considered redundant. Moreover, since DC2AP and DC2DP are used to document patterns and store them in a reuse infrastructure, i.e., APRI, it was considered worth maintaining the same name of the elements used in the two profiles whenever possible.

The DC2AP profile contains elements to control versions of the documented patterns, besides other elements for sharing usage experience. According to Vegi et al. (2012a), these characteristics were incorporated into DC2AP so as to allow new and enhanced versions of the patterns to be proposed with the collaboration of their usage experience. In addition, all versions of these patterns may relate with one another, a resource which enables users to recover the version that best and most efficiently meets their needs. Based on the importance of version control and on the help it provides in the mapping in sub-section 3.2, DC2DP kept the following elements: History and Relation.

After the elements that make up DC2DP had been defined, rules were proposed regarding obligatoriness, occurrence, and type of value of each of the elements proposed. These rules are presented in Figure 3.

Due to space constraints, the semantic description of each element that makes up DC2DP, as well as the details of the application rules, are not presented in this paper, but a detailed specification of the profile can be found at <http://www.dpi.ufv.br/projetos/apri>.

4 DESIGN PATTERN REPOSITORY IN APRI

The goal of the Analysis Patterns Reuse Infrastructure (APRI) is to provide an environment that enables the dissemination and evolution of analysis patterns using an approach targeted to Web Services (Vegi et al., 2012b).

An APRI maintains a repository of analysis patterns documented with the DC2AP metadata profile. Thus, one of the main roles of an APRI is to allow the patterns to be discovered by the users, besides offering tools that allow the patterns to be updated.

This paper extends the APRI structure (Figure 4) by adding a repository of design patterns documented by means of the DC2DP profile. Hence, the analysis patterns may be referenced by design patterns and vice-versa.

A catalog with the design patterns documented by using the DC2DP profile and stored in the APRI repository is available at: <http://www.dpi.ufv.br/projetos/apri>.

The main advantage of the proposed repository, is that, all the design patterns are gathered in the

same place, this way facilitating the retrieving of the patterns and consequently their reuse. Moreover, the proposed metadata profile to document the design patterns use the concept of linked data, which allows a design pattern to create a link between all the other patterns, that can somehow help in its implementation, thereby making a more complete documentation of the pattern.

Elements of DC2DP		
1.Identifier [M][S][UNS]		
2. Title [M][S][St]	2.1 Alternative Title [O][Mu][St]	
3. Classification [O][S][St]		
4. Creator [M][Mu][St]		
5. Subject [M][Mu][St]		
6. Description [M][S][N]	6.1 Problem [M][S][St]	
	6.2 Motivation [M][Mu][St]	6.2.1Example [M][Mu][St]
		6.2.2 Known Uses [M][Mu][St]
7. Date [M][S][N]	7.1 Created [M][S][D]	
	7.2 Modified [Cd][S][D]	
8. Publisher [O][Mu][St]		
9. Contributor [Cd][Mu][St]		
10. Type [M][S][US]		
11. Source [Cd][S][US]		
12. Language [M][S][US]		
13. Relation [Cd][S][N]	13.1 Is version of [Cd][S][UNS]	
	13.2 Is Replaced By [Cd][Mu][UNS]	
	13.3 Replaces [Cd][Mu][UNS]	
	13.4 Is Part of [O][Mu][UNS]	
	13.5 Has Part [O][Mu][UNS]	
	13.6 Related Patterns [O][Mu][UNS]	
	13.7 Is Analyzed With [O][Mu][UNS]	
14. Coverage [O][Mu][St]		
15. Rights [Cd][Mu][US]		
16. History[M][Mu][N]	16.1 Event Date [M][S][D]	
	16.2 Author [M][Mu][St]	
	16.3 Reason [M][S][St]	
	16.4 Changes [Cd][S][St]	
17. Behavior Modelling[O][S][N]	17.1 Sequence Diagram [O][S][U]	
18. Structure Modelling [M][S][N]	18.1 Object Diagram [O][S][U]	
	18.2 Class Diagram [M][S][U]	18.2.1 Class Description [M][S][St]
		18.2.2Relation Description [M][S][St]
19. Sample Code [M][S][N]	19.1 Programming Language [M][Mu][St]	
	19.2 Code [M][Mu][St]	
	19.3 Code Description [M][Mu][St]	
20. Design Guidelines [O][Mu][St]		
21. Consequences [M][S][N]	21.1 Positive [M][Mu][St]	
	21.2 Negative [O][Mu][St]	

Rules' Acronyms		
Obligatoriness	Occurrence	Value Type
[M] Mandatory [O] Optional [Cd] Conditional	[S] Single [Mu] Multiple	[St] String [D] Date [U] URI [N] Null [UNS] URI, number or string [US] URI or string

Figure 3: Elements of the Dublin Core Application Profile to Design Patterns.

The use of the linked data together with the semantic web concept, allow the data that is going to be recovered in the researches, to be more precise and return only what the user really desire, without excess data.

However, the disadvantage of this repository is the need of the user to have access to the internet, so that the data can be retrieved and the patterns visualized.

In order to exemplify the application of the metadata profile proposed in this study, Figure 5 presents a design pattern documented in DC2DP. The design pattern specified is the Singleton, proposed by Gamma et al. (1995). Due to space issues, only the mandatory elements are shown in the description of this pattern.

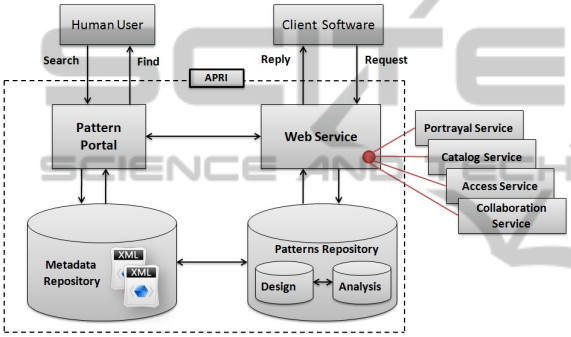

Figure 4: Extension of the Analysis Patterns Reuse Infrastructure (APRI).

5 FINAL CONSIDERATIONS

The DC2DP profile was specified aiming at enabling a more detailed specification of design patterns, since it has been developed specifically to this domain. Through that, this profile must be able to provide users with more complete information, when such information is available, thus helping them to choose the pattern that best aids in carrying out a project and consequent quicker and more effective design production.

With the APRI architecture extension proposed in this paper, the service-oriented infrastructure for cataloging and reusing analysis patterns may be extended and adapted to catalog and reuse design patterns, thus enriching the APRI architecture. This proposal will allow design patterns to be easily found, studied, and reused, also by using web services.

As a future work, an experiment is intended to assess the APRI structure in order to check whether providing these patterns in a repository will in fact help users in their designs. Moreover, an APRI prototype will be created so as to technologically enable the practical application of DC2AP and DC2DP in documenting patterns.

ACKNOWLEDGEMENTS

This project was partially financed by the agencies CAPES, CNPq, FAPEMIG and by the company Funarbe.

REFERENCES

Alur, D.; Crupi, J.; Malks. D., 2003. Core J2EE Patterns: Best Practices and Design Strategies. Sun Microsystems. Palo Alto.

DCMI - Dublin Core Metadata Initiative, 1998. Dublin Core metadata element set, v.1.0: Reference description. Available in: <http://www.dublincore.org/documents/1998/09/dces/>.

Fowler, M., 1997. Analysis Patterns: reusable object models. Addison-Wesley Publishing.

Gamma, E., Helm, R., Johnson, R., Vlissides, J., 1995. Design patterns: elements of reusable object-oriented software. Reading: Addison Wesley Publishing Company.

Pantoquilho, M., Raminhos, R., Araújo, J., 2003. Analysis patterns specifications: filling the gaps. In: *Viking Plop Conference*, Bergen, Norway. p. 169-180.

SOA Pattern, 2013. Available in: <http://soapatterns.org/design_patterns/overview>.

Vegi, L. F. M., Lisboa-Filho, J., Costa, G. L. S, Oliveira, A. P.; Braga, J. L., 2012a. DC2AP: A Dublin Core application profile to analysis patterns. In: *Int. Conf. on Software Engineering and Knowledge Engeneering (SEKE)*, Redwood City, California, USA.

Vegi, L. F. M., Lisboa-Filho, J., Crompvoets, J., 2012b. A machine-processable Dublin Core application profile for analysis patterns to provide linked data. *In: Int. Conf. on Dublin Core and Metadata.* Kuching, Sarawak, Malaysia.

<table>
<tr><td colspan="4">1.Identifier: http://purl.org/apri/metadata/Singleton-v1</td></tr>
<tr><td colspan="4">2. Title: Singleton</td></tr>
<tr><td colspan="4">3. Creator: Erich Gamma, Ralph Johnson, Richard Halm, John Vlissides</td></tr>
<tr><td colspan="4">4. Subject: Single instance, Global variable</td></tr>
<tr><td rowspan="3">5.Description</td><td colspan="3">5.1 Problem: Ensure a class only has one instance, and provide a global point of access to it.</td></tr>
<tr><td colspan="2">5.2 Motivation:
1. It’s important for some classes to have exactly one instance.
2. A global variable makes an object accessible, but it doesn’t keep you from instantiating multiple objects.
3. A better solution is to make the class itself responsible for keeping track of its sole instance. The class can ensure that no other instance can be created, and it can provide a way to access the instance.</td><td>5.2.1Example: 1. Use the Singleton pattern when there must be exactly one instance of a class, and it must be accessible to clients from a well-known access point.
2. When the sole instance should be extensible by subclassing, and clients should be able to use an extended instance without modifying their code.

5.2.2 Known Uses: 1. The InterViews user interface toolkit [LCI+92] uses the Singleton pattern to access the unique instance of its Session and WidgetKit classes, among others.</td></tr>
<tr><td>6. Date:</td><td colspan="3">6.1 Created: 1995</td></tr>
<tr><td colspan="4">7. Type: Design Pattern</td></tr>
<tr><td colspan="4">8. Language: English – EN</td></tr>
<tr><td rowspan="3">9. History</td><td colspan="3">9.1 Event Date: 1995</td></tr>
<tr><td colspan="3">9.2 Author: Erich Gamma, Ralph Johnson, Richard Halm, John Vlissides</td></tr>
<tr><td colspan="3">9.3 Reason: Creation of this design pattern.</td></tr>
<tr><td rowspan="2">10.Structure Modelling</td><td>10.1Class Diagram:
http://purl.org/apri/patterns/Singleton_v1</td><td colspan="2">10.1.1 Class Description: Singleton: defines an Instance operation that lets clients access its unique instance. Instance is a class operation (that is, a class method in Smalltalk and a static member function in C++). May be responsible for creating its own unique instance.</td></tr>
<tr><td colspan="2">10.1.2 Relation Description: Clients access a Singleton instance solely through Singleton's Instance operation.</td></tr>
<tr><td rowspan="2">11. Sample Code</td><td colspan="3">11.1 Programming Language: C++</td></tr>
<tr><td>11.2 Code:
class MazeFactory {
public:
static MazeFactory* Instance();
// existing interface goes here
protected:
MazeFactory();
private:
static MazeFactory* _instance;
};</td><td colspan="2">11.3 Code Description: MazeFactory defines an interface for building different parts of a maze. What's relevant here is that the Maze application needs only one instance of a maze factory, and that instance should be available to code that builds any part of the maze. This is where the Singleton pattern comes in. By making the MazeFactory a singleton, we make the maze object globally accessible without resorting to global variables. We make it a Singleton class in C++ by adding a static Instance operation and a static _instance member to hold the one and only instance. We must also protect the constructor to prevent accidental instantiation, which might lead to more than one instance.</td></tr>
<tr><td rowspan="2">12. Consequences</td><td colspan="3">12.1 Positive: 1. Controlled access to sole instance. Because the Singleton class encapsulates its sole instance, it can have strict control over how and when clients access it.</td></tr>
<tr><td colspan="3">12.2 Negative: Cannot inhibit access your class. Any part of the code to call the method Instance (), because it is static, and have access to the data class.</td></tr>
</table>

Figure 5: Specification of the Singleton design pattern.