

Applications of Extensible Markup Language to Mobile Application Patterns

Hsueh-Ieng Pai

School of Computer Science

McGill University, Montreal

April 2002

A thesis submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements of the degree of Master of Science

Copyright © Hsueh-Ieng Pai, 2002



National Library
of Canada

Acquisitions and
Bibliographic Services

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque nationale
du Canada

Acquisitions et
services bibliographiques

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence

Our file Notre référence

The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

0-612-78934-9

Abstract

Mobile applications provide services that can benefit various sectors of the society. It is imperative that the realization of mobile applications be well-planned and based on standards. The interplay of Extensible Markup Language (XML)-based technologies, software engineering principles, and “best practices” formalized as patterns provides such a systematic approach. This thesis formulates a rigorous classification of mobile application patterns into four categories: architecture, process, product, and usage. To express the patterns in a universal manner, an XML-based pattern notation, the Mobile Application Patterns Markup Language (MAPML), is introduced. Based on a requirements analysis and design description, the specification of MAPML is given. MAPML is equipped with a collection of tools that includes formal grammars for MAPML, and solutions for authoring, processing, and presenting MAPML documents on the Web. The thesis concludes with a discussion of the current state of work and directions for future improvement.

Abstrait

Les nouveaux logiciels dédiés à la mobilité, permettent à plusieurs secteurs de la société d'en bénéficier. Il est impératif que la réalisation de ces nouveaux logiciels soit bien organisée et qu'elle respecte une norme standardisée. Le rôle que joue les technologies basées sur le langage XML (Extensible Markup Language), les principes d'ingénieries en informatique, ainsi que les meilleures réalisations formalisées sous formes de modèles, fournissent une approche systématique. Cette thèse dresse une classification rigoureuse des différentes configurations des applications mobiles en quatre catégories: architecture, processus, produit, et utilisation. Pour expliquer cette configuration d'une façon universelle, on introduit la notation de la configuration du langage XML, ainsi que la configuration des applications mobiles du langage XML (Mobile Application Patterns Markup Language (MAPML)). Basé sur les conditions d'analyse et la description de la conception, on donne aussi le cahier de charges du langage MAPML. Le MAPML est équipé d'outils pour l'édition, le traitement ainsi que la présentation de documents MAPML sur le Web. La thèse se conclut par une discussion sur l'état du travail courant et donne des directives pour une amélioration future.

Acknowledgements

I would like to give my sincere gratitude to my thesis supervisor, Dr. Monty Newborn, who has supported my work for all these years with patience and kindness. He has given me valuable suggestions, remarks, and clear guidelines that contribute to the success of this thesis.

I am also grateful to Mr. Pankaj Kamthan, Department of Computer Science, Concordia University, for his illuminating discussions and support. Mr. Kamthan has introduced me to the world of Extensible Markup Language. In addition, he has provided me with pointers to many useful resources, and has given me critical feedbacks on the thesis throughout the years. He has always been a source of guidance and inspiration.

I would also like to thank my friends, Richard Francoeur and Omar Meghari, for translating the abstract into French.

Last but not least, I would like to thank my family for their love and support.

Table of Contents

| | |
|----------------------|----|
| List of Figures..... | vi |
|----------------------|----|

| | |
|----------------------|-----|
| List of Tables | vii |
|----------------------|-----|

| | |
|------------------------------|---|
| Chapter 1 Introduction | 1 |
|------------------------------|---|

| | |
|---|---|
| 1.1. Limitations of Current Approaches Towards Realizing Mobile Applications | 1 |
| 1.2. Significance of a Systematic Approach Towards Realizing Mobile Applications..... | 2 |

| | |
|---|---|
| Chapter 2 Mobile Applications, XML, and Patterns..... | 5 |
|---|---|

| | |
|--|----|
| 2.1. Significance of Mobile Applications..... | 5 |
| 2.2. Extensible Markup Language (XML) in Mobile Applications | 7 |
| 2.2.1. Motivation for Use of XML in Mobile Applications: Business Perspective | 7 |
| 2.2.2. Motivation for Use of XML in Mobile Applications: Consumer Perspective | 13 |
| 2.3. On the Definition of Patterns..... | 14 |
| 2.3.1. On the Significance of Patterns | 15 |
| 2.4. Classification of Patterns in Mobile Environment | 17 |
| 2.4.1. Classification of Mobile Patterns | 17 |
| 2.5. Classification of Mobile Application Patterns | 20 |
| 2.5.1. Relationship Across Patterns..... | 20 |
| 2.5.2. Mobile Application Process Patterns | 21 |
| 2.5.3. Mobile Application Architecture Patterns..... | 22 |
| 2.5.4. Mobile Application Product Patterns | 23 |
| 2.5.5. Mobile Application Usage Patterns..... | 31 |
| 2.5.6. Relationship to General Pattern Taxonomies..... | 31 |
| 2.6. Pattern Language | 32 |
| 2.6.1. Limitations of Traditional Pattern Language Notation | 32 |
| 2.6.2. The Significance of XML Representation of Patterns and the Motivation for MAPML | 34 |
| 2.6.3. MAPML and XML..... | 35 |
| 2.7. Summary..... | 35 |

| | |
|---|----|
| Chapter 3 MAPML Requirements and Design | 36 |
|---|----|

| | |
|-------------------------------------|----|
| 3.1. MAPML Use Cases | 36 |
| 3.2. MAPML Requirements | 39 |
| 3.3. MAPML Design Description | 45 |
| 3.4. Summary..... | 50 |

| | |
|--|------------|
| Chapter 4 MAPML Specification | 51 |
| 4.1. MAPML Modules and Element Definitions | 51 |
| 4.2. Properties of MAPML Elements and Attributes | 54 |
| 4.2.1. MAPML Data Types | 54 |
| 4.2.2. Enumeration | 54 |
| 4.3. Association Module | 55 |
| 4.4. Meta-Information Module | 58 |
| 4.5. Problem Module | 65 |
| 4.6. Solution Module | 67 |
| 4.7. Structure Module | 74 |
| 4.8. MAPML Attribute Definitions | 76 |
| 4.9. MAPML Identification | 81 |
| 4.9.1. MAPML Internet Media (MIME) Type | 81 |
| 4.9.2. MAPML Namespace | 81 |
| 4.9.3. MAPML Filename Extension | 82 |
| 4.10. MAPML Conformance | 82 |
| 4.10.1. MAPML Document Conformance | 82 |
| 4.10.2. MAPML Processor Conformance | 83 |
| 4.11. Summary | 83 |
| Chapter 5 MAPML Utilities (MAPML-UTIL) | 84 |
| 5.1. MAPML Grammars | 84 |
| 5.1.1. EBNF Implementation for MAPML | 85 |
| 5.1.2. XML DTD Implementation for MAPML | 85 |
| 5.1.3. XML Schema Implementation for MAPML | 86 |
| 5.1.4. RELAX NG Implementation for MAPML | 86 |
| 5.2. Authoring MAPML Documents | 87 |
| 5.2.1. MAPML with XEENA | 87 |
| 5.2.2. MAPML with VIM | 88 |
| 5.3. Processing MAPML Documents | 90 |
| 5.3.1. XML APIs and MAPML | 90 |
| 5.3.2. MAPML and Data Binding | 90 |
| 5.4. Presenting MAPML Documents | 93 |
| 5.4.1. Direct Presentation of MAPML Documents | 94 |
| 5.4.2. Indirect Presentation of MAPML Documents Transformed to XHTML | 95 |
| 5.4.3. Associating Style Sheets with MAPML-Related Documents | 96 |
| 5.5. Summary | 98 |
| Chapter 6 Conclusion | 99 |
| Bibliography | 102 |
| Appendix A | 108 |

List of Figures

| | |
|--|-----|
| Figure 1.1. The "Quadrangle" of Relationships between Mobile Application, Extensible Markup Language, Software Engineering, and Patterns. | 3 |
| Figure 1.2. The MAPML Environment. | 4 |
| Figure 2.1. Mobile Applications in a Classification of Pervasive Internet Applications. .. | 5 |
| Figure 2.2. A Panorama of Mobile Applications. | 6 |
| Figure 2.3. A Snapshot of XML Vocabulary Spectrum. | 8 |
| Figure 2.4. The XML Document/Data Dichotomy. | 12 |
| Figure 2.5. The Pattern Anatomy and Cycle. | 14 |
| Figure 2.6. Classification of and Inter-Relationship among Mobile Patterns. | 17 |
| Figure 2.7. Mobile System with Visual Interface Architecture Pattern. | 19 |
| Figure 2.8. Mobile System with Speech Interface Architecture Pattern. | 20 |
| Figure 2.9. Classification of and Inter-Relationship among Mobile Application Patterns. | 21 |
| Figure 2.10. Mobile Application Architecture Patterns. | 22 |
| Figure 2.11. Combination (Sequential and Hierarchical) Architecture Pattern. | 22 |
| Figure 2.12. A Hierarchical Classification of Product Patterns in Mobile Applications. .. | 23 |
| Figure 3.1. MAPML Transformation Use Case Diagram. | 38 |
| Figure 3.2. MAPML Transformation Use Case Scenario. | 38 |
| Figure 3.3. MAPML Modularization Process. | 47 |
| Figure 4.1. Structure of MAPML. | 53 |
| Figure 5.1. MAPML Family of Utilities (MAPML-UTIL). | 84 |
| Figure 5.2. Xena-MAPML DTD Interface. | 88 |
| Figure 5.3. MAPML Syntax File in the VIM Interface. | 89 |
| Figure 5.4. MAPML Data Binding Process. | 92 |
| Figure 5.5. Marshalling and Un-Marshalling MAPML Documents. | 92 |
| Figure 5.6. The Two Approaches for Presentation of MAPML Documents. | 93 |
| Figure 5.7. Internal Structure of a CSS Style Sheet. | 94 |
| Figure 5.8. The MAPML to XHTML via XSLT Process. | 96 |
| Figure 6.1. Role of XML Transformations in Delivering and Presenting [XML] Documents in Multiple-Environments. | 101 |
| Figure 6.2. Transformations of XML Book Vocabularies to E-Book Vocabularies. | 101 |

List of Tables

| | | |
|-------------|---|----|
| Table 2.1. | Initial Product Patterns of XML-Based Mobile Markup Language Elements with Example Implementation(s)..... | 24 |
| Table 2.2. | Non-Autonomous Intermediate Product Patterns with Example Implementation(s)..... | 27 |
| Table 2.3. | Autonomous Intermediate Product Patterns with Example Implementation(s)..... | 27 |
| Table 3.1. | Examples of MAPML Use Cases..... | 37 |
| Table 4.1. | MAPML Modules..... | 52 |
| Table 4.2. | The class Element..... | 56 |
| Table 4.3. | The link Element..... | 56 |
| Table 4.4. | The pattern.related Element..... | 57 |
| Table 4.5. | The reference Element..... | 58 |
| Table 4.6. | The abstract Element..... | 58 |
| Table 4.7. | The author Element..... | 59 |
| Table 4.8. | The caption Element..... | 60 |
| Table 4.9. | The date Element..... | 60 |
| Table 4.10. | The description Element..... | 61 |
| Table 4.11. | The keyword Element..... | 61 |
| Table 4.12. | The license Element..... | 62 |
| Table 4.13. | The metadata Element..... | 63 |
| Table 4.14. | The name Element..... | 64 |
| Table 4.15. | The term Element..... | 65 |
| Table 4.16. | The title Element..... | 65 |
| Table 4.17. | The constraint Element..... | 66 |
| Table 4.18. | The context Element..... | 67 |
| Table 4.19. | The problem Element..... | 67 |
| Table 4.20. | The consequence Element..... | 68 |
| Table 4.21. | The implementation Element..... | 69 |
| Table 4.22. | The object Element..... | 70 |
| Table 4.23. | The rationale Element..... | 71 |
| Table 4.24. | The scenario Element..... | 72 |
| Table 4.25. | The solution Element..... | 73 |
| Table 4.26. | The strategy Element..... | 73 |
| Table 4.27. | The structure Element..... | 74 |
| Table 4.28. | The body Element..... | 75 |
| Table 4.29. | The head Element..... | 75 |
| Table 4.30. | The mapml Element..... | 76 |
| Table 4.31. | The pattern Element..... | 76 |
| Table 4.32. | The alternate Attribute..... | 77 |
| Table 4.33. | The event Attribute..... | 77 |
| Table 4.34. | The id Attribute..... | 78 |
| Table 4.35. | The impact Attribute..... | 78 |

| | |
|--|----|
| Table 4.36. The media-type Attribute..... | 79 |
| Table 4.37. The object-type Attribute..... | 79 |
| Table 4.38. The relation Attribute..... | 80 |
| Table 4.39. The term-type Attribute | 80 |
| Table 4.40. The uri Attribute | 80 |
| Table 4.41. The version Attribute | 81 |

Chapter 1

Introduction

In the last few years, there have been significant changes in the way digital information is disseminated, accessed, and processed. The proliferation of mobile technologies has been exponential, and both support and use for it is gaining momentum. This, as recent surveys indicate [14], will have a significant impact on enterprise-wide information systems and pay-per-use public services. At the heart of these facilities are mobile applications.

Currently, the technology for engineering mobile applications is mature. However, there seems to be a lack of methodical approach in doing so. This thesis is an effort to fill that gap by focusing on both functional and non-functional aspects in realizing mobile applications.

1.1. Limitations of Current Approaches Towards Realizing Mobile Applications

The current approaches for creating mobile applications have the following major limitations:

- **Orientation Towards Presentation.** The development of most applications is primarily focused on implementation, with the use of a few presentation-oriented markup languages such as the Wireless Markup Language (WML) [67] and the Compact HTML (cHTML) [46]. One of the major drawbacks of this approach is data management. Presentation formats are less amenable to change. In particular, it is very difficult to extract useful data for indexing and searching, and to make timely adjustments to support multiple devices with diverse configurations.
- **Lack of Process.** The approach to create applications seems to be ad-hoc with little emphasis on a methodology. In general, although this approach may work on a small-scale, it can lead to various problems on a large-scale. For example, it

becomes difficult to trace the evolution of the system, particularly when the development team changes.

- **Mixture of Responsibilities.** There is often no clear division between semantically different tasks. For example, the use of markup languages tends to mix structure, presentation, and logic. This high coupling makes the system difficult to manage.

It has been pointed out [40] that many companies have failed due to unsound business practices and inadequate systems. There were various problems with these companies' systems: were unable to handle the volume of business (scalability issues), were unable to assure customer privacy and security (trust issues), did not maintain consistently high-quality service (quality issues), were inconvenient to use or access (usability issues), and/or did not provide customers with reasonable response times (performance issues). The issues of usability and performance become particularly inflated in a mobile environment due to constrained capabilities of the devices and the nomadicity of the user who may be accessing time-critical information.

1.2. Significance of a Systematic Approach Towards Realizing Mobile Applications

The problems mentioned in the above section have underscored the importance of using a systematic approach in the development of successful mobile applications. To do that, one needs to take into consideration the most appropriate choices for both the product (the mobile application) and the process. One way of doing so, as this thesis proposes, is to use the Extensible Markup Language (XML) [55] for the product, and to apply established principles and practices of software engineering in the process. We also emphasize that both application of XML (for creation of the product) and of software engineering (for creation of the process) can benefit from each other. To make optimal use of XML and software engineering in mobile applications, one must take into account thoroughly-tested and established best practices based on expert knowledge and experience, that is, patterns. Figure 1.1 illustrates the interdependency of mobile application, XML, software engineering, and patterns.

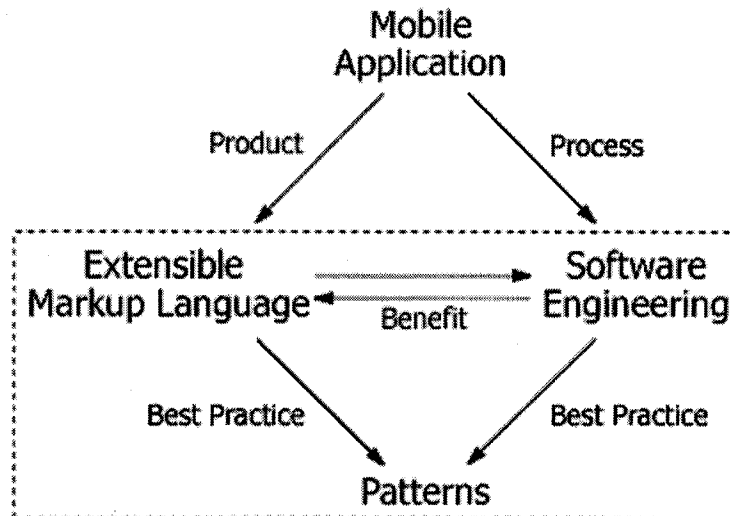


Figure 1.1. The "Quadrangle" of Relationships between Mobile Application, Extensible Markup Language, Software Engineering, and Patterns.

For a consistent representation and machine-to-machine interchange, patterns need to follow a notation. This thesis introduces an XML-based formal notation for mobile application patterns, the Mobile Application Patterns Markup Language (MAPML), and builds a set of supporting tools for MAPML.

The thesis is organized as follows. Chapter 2 functions as the domain analysis for MAPML. It provides the background on patterns necessary for the definition of MAPML. Moreover, it discusses the advantages that XML offers in general, and to mobile applications, in particular. Finally, a classification of mobile patterns that addresses both visual and speech modalities is provided. Chapter 3 discusses the MAPML use cases, which are then used to elicit MAPML requirements. The requirements in turn lead to design issues and descriptions. This sets the stage for a formal definition of MAPML. Chapter 4 provides the complete MAPML specification with the details of its syntax and semantics. Chapter 5 includes a set of supporting utilities for MAPML users. The MAPML environment that presented from chapters 2 to 5 is shown in Figure 1.2. Finally, Chapter 6 concludes with remarks on the current state of MAPML and directions of evolution.

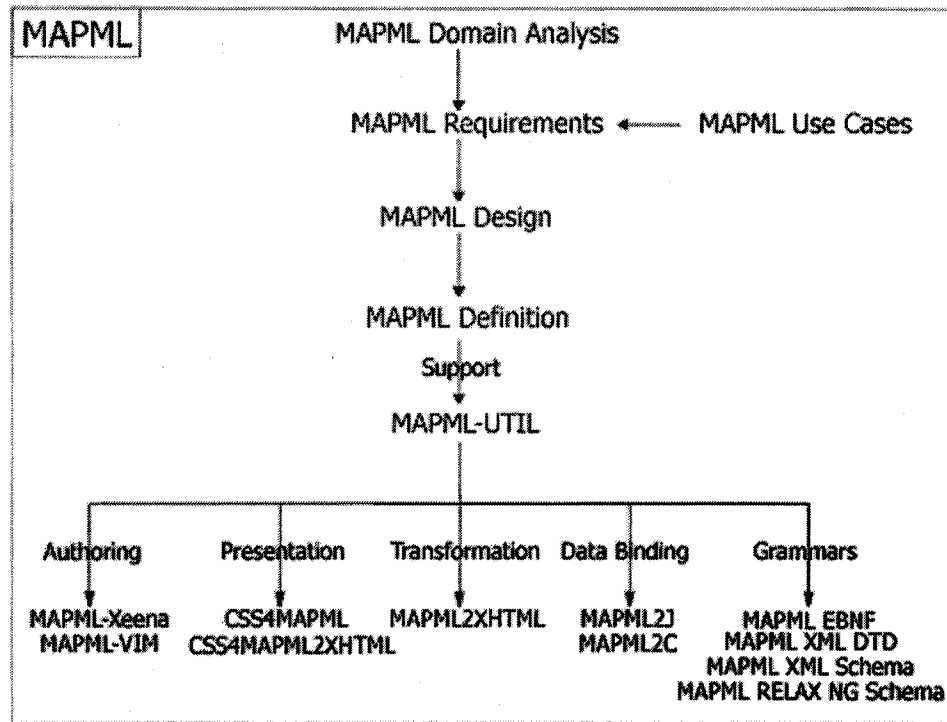


Figure 1.2. The MAPML Environment.

Chapter 2

Mobile Applications, XML, and Patterns

This chapter provides the motivation and background on mobile applications, XML, and patterns. Section 2.1 discusses the significance of mobile applications to both business and consumer. The advantages of using XML in mobile applications are discussed in Section 2.2. Section 2.3 gives the definition of pattern, and elicits several characteristics of patterns that make them suitable for mobile application development. Section 2.4 classifies patterns that occur in mobile environment in general, while Section 2.5 classifies patterns that occur in mobile applications in particular. Finally, Section 2.6 introduces the notation currently used to express patterns, points out its limitations in a mobile application setting, and outlines the advantages that XML offers as a form for representing patterns.

2.1. Significance of Mobile Applications

Mobile applications are characterized by the use of resource-constrained devices to access various services where the user-location is often not fixed. These applications are part of the general framework of pervasive computing applications (Figure 2.1).

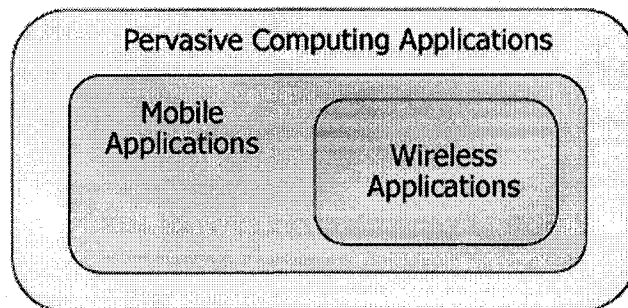


Figure 2.1. Mobile Applications in a Classification of Pervasive Internet Applications.

Figure 2.2 shows a set of possible mobile applications. Location-based services [64] provide access to information depending on user's geographical location. Examples

include map servers that provide road maps (and local weather/traffic conditions) of the region in real-time to a traveler who is driving through. Mobile office allows access to corporate information by employees on the road (or in the air). Examples include update of inventories, price, and client lists by sales people. Syndication services provide information that may be of interest to the user who has subscribed to the service. Examples include news, advertisements, stock quotes, and so on. Mobile commerce refers to electronic transactions for the exchange of goods or services for monetary consideration via mobile devices. In other words, mobile commerce is electronic commerce using a mobile device such as a mobile phone or a personal digital assistant (PDA). Electronic books (E-Books) [19] provide small interactive books that can be accessed via mobile devices such as a PDA or a special-purpose digital book reader. Examples include travel guides with information that is navigable and searchable, and incorporates media clips of sites that are popular tourist attractions.

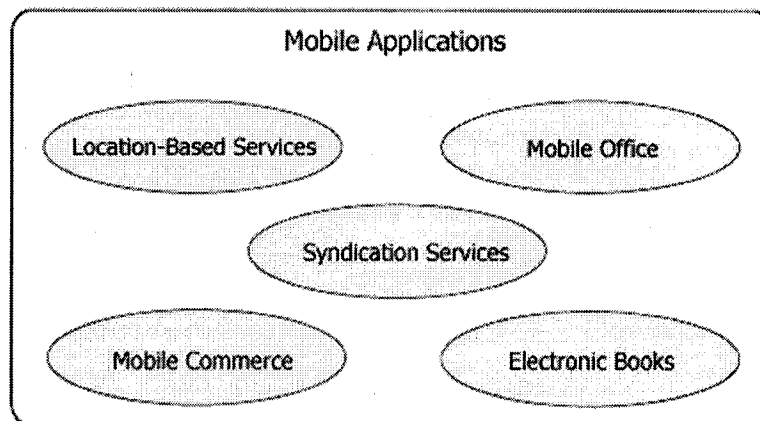


Figure 2.2. A Panorama of Mobile Applications.

There are both similarities and differences between the mobile and Web environments [48]. Mobile applications leverage on the existing infrastructure of the Web for the server-side technologies, the network (Internet), the resource addressing schemes, and the data communication protocols (TCP/IP). They differ mainly on the client-side, where the accessing devices have limited resources (of memory, size, weight, range of connectivity, and processing power), and the user location can be dynamic. These

similarities and differences come into play during the transition of an existing knowledge and resource-base from one environment to another.

2.2. Extensible Markup Language (XML) in Mobile Applications

The purpose of markup is to associate “context” with data so that machine processing becomes more “intelligent.” XML is a meta-language that provides the direction for syntax for markup languages.

There are several characteristics of XML that make it the de facto standard of choice for data interchange. The advantages of using XML for mobile applications, both from business and consumer's perspective, are discussed in detail in the following sections.

2.2.1. Motivation for Use of XML in Mobile Applications: Business Perspective

This section describes the benefits that XML offers to a business creating mobile applications.

Standardization

It is important for both B2B and B2C e-commerce to standardize the information representation. XML is a system-independent, vendor-independent, and application-independent open standard that is itself based on standards such as Unicode for character encoding, SGML for syntax, EBNF for grammar, and URIs for name identifiers. Hence, from device manufactures' point of view, no license will be needed for producing devices that support XML. In addition, from application developer's view, as long as the information is encoded in XML, it can be displayed on any wireless device that supports XML without having to re-write the document for displaying on each type/variety of device.

XML is a W3C Recommendation. XML is therefore deemed "stable" and ready for widespread deployment. Standardization of XML creates a common ground for business communication.

Business-Data-Sensitivity

XML, as a meta-language, provides a standard framework to create business-oriented markup vocabularies. It is now possible to design industry-specific data formats with atomic level of granularity (coarse-graininess) that are intimately sensitive to the business data context. Several vocabularies targeting specific industries have come into existence in the last 2 to 3 years. Figure 2.3 provides a glimpse of the application domains that have been "captured" in XML.

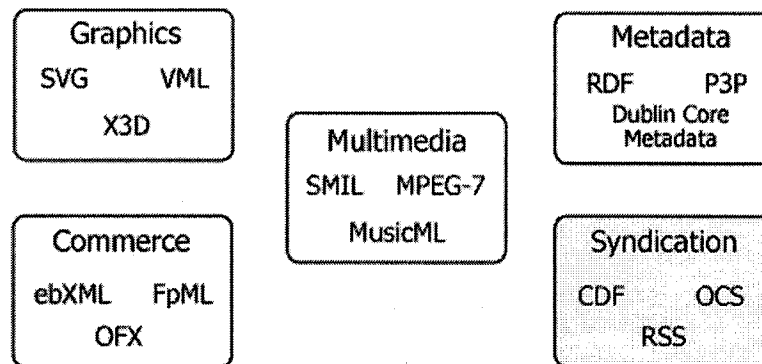


Figure 2.3. A Snapshot of XML Vocabulary Spectrum.

Content Management

As information increases, it becomes necessary to have a system in place to archive, retrieve, and query it with ease and efficiency. There are three factors that contribute to advantages of using XML towards content management:

1. **Timelessness.** Proprietary or binary data formats restrict the range of usability of a content management system. XML provides data longevity, a crucial factor for a long-term data management strategy.
2. **Reusability.** For data reusability, it is important that what is represented (content) be separated from the way it is rendered (presentation).

3. **Transformability.** The fact that XML documents are in plain text and structured leads to prospects of customized data delivery to multiple devices with different user requirements.

By mixing structure and presentation, a document becomes "inclined" towards a specific-device and hence becomes inflexible (for reusability). XML separates structure from presentation. Hence, the same XML document can be transformed to produce different types of outputs for different mobile devices without the need to modify the original content. For the application developers, this means that they no longer need to keep different versions of the same document for displaying on mobile devices that have different capabilities. In addition, when modifications are required, only the original version of the content needs to be edited before republishing to the various target devices. This leads to efficiency and ease-of-maintainability, without the inherent problems of version control and the effort required in making modifications in medium-specific document versions. Hence, the developers can concentrate on authoring rather than formatting, and thus be more productive.

Data Storage

XML content can be stored in Tables, as Documents, as Objects, and Natively, providing various benefits (depending on the requirements for scalability, stability, and reliability). As a result, various XML retrieval (query languages) and archival (database) schemes are emerging.

Data Interchange

XML allows interoperable data interchange between distributed networks. This is especially important when the business is conducted in a wireless environment. For example, for employees who need to work on the field (such as salesmen), the most up-to-date information (such as the number of product in stock) can be retrieved from the company database by simply accessing the company's Web site from a mobile device. There is no need to install any proprietary software on the mobile device in order to view the data represented in company's internal format (which is very important since the

mobile devices have limited capability, including the memory). All that is required is that the company develops the mapping to transform the internal format used by the back-end systems into XML documents.

Internationalization

A major advantage of conducting business on the Web is that it broadens the customer-base towards globalization, without the necessity of having physical office locations. This philosophy is only strengthened in the mobile arena. However, in order to communicate, a business must still "speak" the language of the region in context.

With the Unicode support in XML, mobile sites can be multilingual. This allows businesses to deploy location-sensitive applications in the language-of-choice of customers.

Data Conformance

As data interchange becomes more and more important in a distributed networking environment, the need for data quality assurance increases. Lack of systematic techniques for validation can lead to disastrous consequences, such as corruption of one business's data entering another business's database, crash of the user client, abrupt delays in transit, and so on.

XML provides a powerful mechanism of structural, syntactical, and data type validation via XML Schema Languages. XML grammars can function as "legal contracts" between businesses. One business can develop more trust in other business's data if it is known to be validated. Such trust is particularly crucial in a mobile commerce arena where data from one business has to often go through an intermediary (middleware) for adaptation (transcoding). For example, a proxy that is receiving news data (say, in NewsML) from a broadcasting company to be transformed to a mobile markup language (say, WML) would like to be assured that the data is uncorrupted before it enters its adaptation engines.

Interoperability

When businesses exchange data, it is imperative that when one (human, machine) "talks", other "listens". Universal interoperability is a long-term ideal goal that involves cooperation between several parties: operating systems, data processing programs, and so on. XML provides the first step toward data interoperability by creating a standard framework on which businesses can agree upon.

Cost Effectiveness

When a business deploys a data format in its systems, it is important that the data has a long life span. Re-engineering data to conform to different platforms or software can be resource intensive and expensive. An XML-based framework is a long-term investment (compared to a proprietary data format/corresponding system). XML also eliminates the need for developing multiple versions of the same document to be displayed on mobile and non-mobile devices, thereby reducing the cost.

Processability

There are three factors that contribute to advantages of XML towards processability:

1. **Universality.** Any plain text editor can be used to author XML documents. Any pattern-matching languages, including existing shell and scripting languages, can be used to process XML documents.
2. **XML Duality: Process as Data, Present as Document.** There are dual roles played by XML: Human-to-Machine (Presentation) and Machine-to-Machine (Data Interchange). XML, as data, is highly structured and machine-centric (oriented towards exchange). XML, as document, is less structured and human-centric (oriented towards readability). Figure 2.4 illustrates the XML Document/Data Dichotomy model.

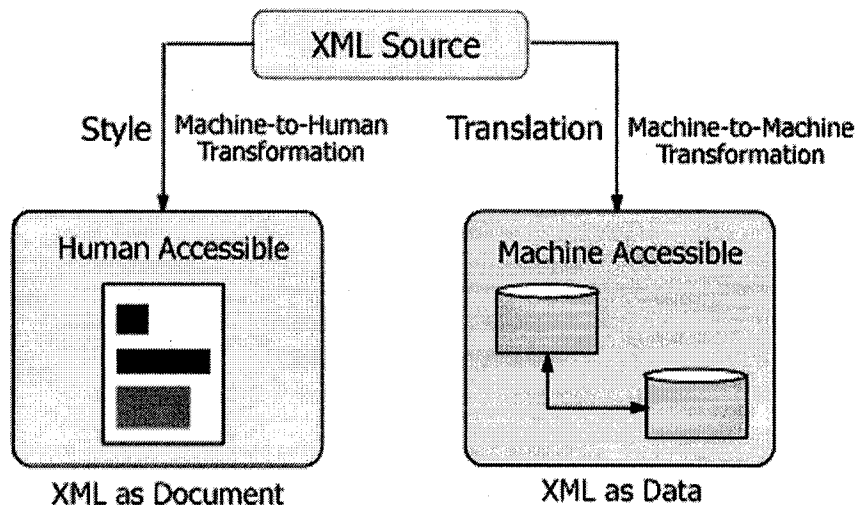


Figure 2.4. The XML Document/Data Dichotomy.

3. **APIs and Software.** There has been an increasing base for standard APIs (with bindings to a variety of widely-used programming languages) and (open source) software for XML processing. This levels the playing field for independent developers and software engineers. The growing pool of (widely and freely) available software allows Small and Medium Size Businesses (SMEs) to become participants. Mobile applications can leverage on and learn a lot from that experience.

Since XML separates structure from presentation, automated machine processing (for which presentation is irrelevant) becomes possible. (This is a major improvement over HTML that allows the mixture of structure and presentation.) The content marked-up in XML can be intelligently searched, manipulated, and rearranged. For example, with the help of Composite Capabilities/Preference Profiles (CC/PP) [59] and its implementations [66] that describe device capabilities and user preferences, the business can now serve the customer with personalized information. Instead of filling up the whole screen with some irrelevant information, only what's interested to the user will be displayed. In addition, the display will be customized to the mobile device used by the user. This is especially important since the mobile devices generally have low bandwidth, limited screen size, and a variety of display capabilities.

2.2.2. Motivation for Use of XML in Mobile Applications: Consumer Perspective

This section describes the benefits that XML offers to a consumer (user) accessing mobile applications.

Personalization

When marked in an XML format, "smart" agents can customize the information to be viewed on the mobile device according to the preferences of the end user and the capabilities of the client device. Hence, the user could receive personalized information on the wireless device with appropriate display format. This means that, instead of having to scroll down through a lengthy document on small mobile device screen, only the limited and relevant information is shown to the user. In addition, since the size of the personalized document is smaller than the complete document, the time that the user has to wait for the information loading is lessened.

Accessibility

The Web is increasingly being accessed by customers using a variety of devices with different screen capabilities and computing power - personal digital assistants, cellular phones, and so on. XML, being platform-independent, scales well to provide information on these devices without the need for the user to make adjustments at his/her end. So, the user is able to access the same mobile site anywhere, any time, and using any device, no matter if it is text-based, graphics-based, or voice-based.

Interactivity

XML has been used to create more flexible user interface components than that have been previously possible. With structure, presentation, and logic de-coupled, it is now possible to develop neutral mechanisms for document models (artifacts of how a document is structured), to formulate standard interface-independent event models (artifacts of how a events are associated with document objects), and to create programming language bindings for them.

2.3. On the Definition of Patterns

A pattern [4] expresses a relationship between a certain context, a problem, and a solution. A **context** is the environment, situation, or interrelated conditions within the scope of which something exists. A **problem** is an issue that needs to be investigated and resolved, and is typically constrained by the context in which it occurs. The **solution** is a response to the problem in a context that helps resolving the issue(s).

The most important defining characteristics of a pattern are: **identification** (of the problem in context and with imposed constraints), **existence** (of the solution), **recurrence** (of the problem), **invariance** (of aspects of the solution), **practicality** (of the solution, which needs to strike a balance between **optimality** and **objectivity**), and **communicability** (of the problem and the process of arriving at the solution to the user). Some of these characteristics are illustrated in Figure 2.5.

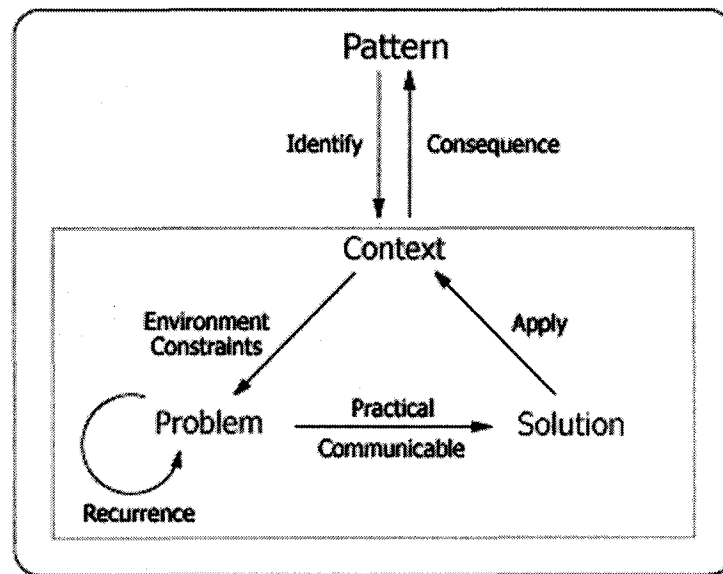


Figure 2.5. The Pattern Anatomy and Cycle.

The concept of patterns was originated in the late 1970's [1] in the civil engineering and urban planning domains as an approach to design buildings, roadways, and towns. The notion of patterns was introduced to the object-oriented software engineering community [16] in the late 1980's. The view taken by this thesis is that, if

patterns can describe aspects of software, they can also be used to describe process, architecture, components, and usage of mobile applications.

2.3.1. On the Significance of Patterns

There are several features that make patterns useful:

Formality

Patterns are formal. There have been several efforts [20, 44] that provide strategies and suggest guidelines for Web application design. However, one of the limitations of these guidelines is that they make broad use of natural language for description, and hence are less formal and vague. Frequently Asked Questions (FAQs) also help answer user questions but they are usually focused on a single topic, often specific to a technology, and rarely provide reasoning for their answers. Patterns are more formal in their approach and exist at a higher level of abstraction than the strategies or guidelines. Patterns offer various advantages over guidelines [18] and are anticipated to play an essential role in information technology [20]. Still, patterns do not attempt to necessarily replace the FAQs, strategies, or guidelines in every manner. Rather, they should be considered as a key complement to the overall initiative of a business application realization.

Nomenclature

Patterns (and therefore the concepts they represent) are assigned names. This expedites further discussion, analysis, and reference of previously internalized concepts.

Practicality

Patterns provide practical "ready-to-go" solutions. A pattern describes "good" practical solutions to a common problem within a certain context, by describing the invariant aspects of all those solutions. Given a problem, patterns include a compact, focused, complete, and straightforward way of describing a solution. Since they provide the consequences of applying that solution, the user can decide and act upon in a timely manner if the solution is applicable to his/her situation.

Experience

Patterns form an "expert" system in practice. Patterns, when well-defined and coordinated, are more than a mere static disjoint "collections" of recipes. Patterns are tried-and-tested ways to deal with problems that recur. It is expected that those who have experience in a particular field of knowledge will have internalized certain solutions to these problems. As a result, they recognize a problem to be solved and know which solution to apply in the particular situation. A pattern describes this internalized expert knowledge and states the problem, context, and solution, so that others with less experience can benefit from this knowledge. In this sense, patterns themselves can be considered as a "smart FAQ" or an "expert system" that encapsulates the knowledge and experience of the author. This enables them to be used as a **knowledge base**.

Re-Usability

A pattern presents a higher-level view of the same problem inflicting often multiple industries and provides a solution for it. It can also connect to other patterns in existence (in the same or other catalogs) for whole or in part of its solution (inheritance). Patterns thus encourage re-use.

Abstract, Modular Framework

Complex problems are often composed of several steps that need to be dealt with independently and then combined to arrive at a solution. Patterns represent these steps at a high-level via "intelligent" distribution and allocation of responsibilities. They provide a framework that works in unison to fulfill a given task.

Community

Patterns help a broad community. Patterns communicate solutions to a community of architects, designers, and engineers, who make use of them at different levels and for different purposes. The goal of the pattern community is to build a re-usable body of knowledge to support design and development in general.

2.4. Classification of Patterns in Mobile Environment

This section presents a classification scheme for patterns in a mobile environment, specifically that are related to mobile applications. These patterns can be viewed as a group of reusable assets that can help expediting the development of mobile applications. We call our mobile pattern classification scheme MAPCLASS. Software engineering principles that form the basis of the classification are separation of concerns (separation into parts -- abstraction, modularity), incrementality, generality, and anticipation of change.

2.4.1. Classification of Mobile Patterns

At the most basic level, mobile patterns can be classified into three classes: Mobile Business Patterns, Mobile System Architecture Patterns and Mobile Application Patterns. The requirements of the Mobile Business will usually determine the Mobile System Architecture and the Mobile Application. Figure 2.6 illustrates the hierarchy and dependency of these patterns.

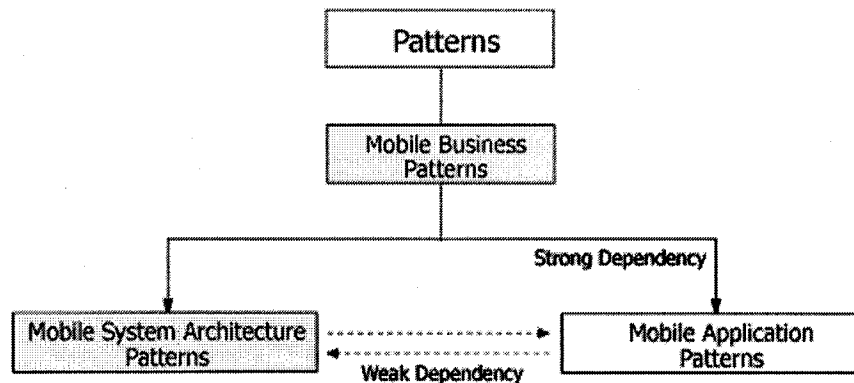


Figure 2.6. Classification of and Inter-Relationship among Mobile Patterns.

2.4.1.1. Mobile Business Patterns

The Mobile Business Patterns encapsulates the different processes followed to successfully realize the business solution. Identification of these is necessary as they

provide directions and constraints that lead to the completion of business objectives and goals.

2.4.1.1.1 Relationship to IBM E-Business Patterns

The IBM Patterns for E-Business [20] represent common e-business problems and are classified into six categories.

1. Business Patterns identify and describe the most commonly observed interactions between the participants in an e-business solution (Users, Businesses, and Data) and are the fundamental building blocks of most e-business solutions. Given a business's requirements, an appropriate Business pattern can be chosen for that set of requirements, and used as a "stepping-stone" for further analysis.
2. Integration Patterns do not themselves automate specific business problems. Instead, combining multiple Business Patterns together to build advanced e-business applications, or to make Composite Patterns feasible by allowing the integration of two or more Business patterns. For example, integrating the Self-Service Pattern with the Information Aggregation Pattern to improve the personalization of a customer self-support system.
3. Composite Patterns are combinations of Business Patterns and Integration Patterns that have themselves recurrently been deployed to solve the problems of businesses across a wide range of industries.
4. Custom Designs, like Composite Patterns, combine Business Patterns and Integration Patterns to create e-business applications to solve the e-business problems of one specific company, or perhaps several enterprises with similar problems. However, Custom Design Patterns do not meet the status of a Composite Pattern, and do not give as great a reassurance of reusability, because they have not been "recurrently employed to solve the problems of businesses across a wide range of industries."
5. Application Patterns are driven by the customer requirements and describe the shape of the e-business application. They are chosen after a Business Pattern, Integration Pattern, or Composite Pattern is selected. Application Patterns

describe how the data and logic of the application are partitioned into tiers and how these tiers interact. The choice of Application Pattern typically leads to an underlying Runtime Pattern.

6. Runtime Patterns are also driven by the customer requirements and describe the supporting runtime needed to build the e-business application.

These basic models are themselves further classified into sub-categories, and eventually mapped onto products. It is recommended that they be applied in a certain order, and used together with guidelines appropriate to the respective business, to arrive at an e-business solution successfully.

The IBM Patterns for E-Business have also been adapted to the context of mobile business. We therefore adopt (and thus re-use) the classification of IBM Patterns for E-Business as our Mobile Business Patterns. The IBM Patterns for E-Business are, for large part, restricted to suggesting proprietary solutions and products developed by IBM and its satellite companies and/or affiliates (IBM WebSphere, IBM Edge Server, IBM DB2, Lotus Domino).

2.4.1.2. Mobile System Architecture Patterns

Mobile System Architecture Patterns reflect the system environment in which the mobile application operates. The two patterns that we have identified are Visual Interface Architecture Pattern and Speech Interface Architecture Pattern (Figures 2.7 and 2.8).

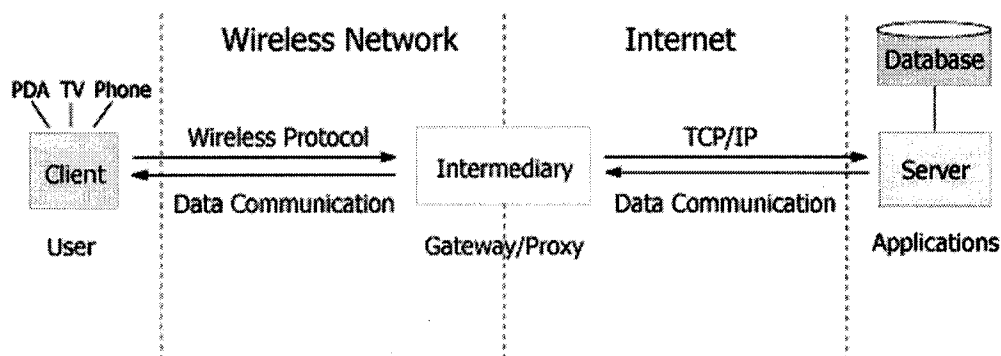


Figure 2.7. Mobile System with Visual Interface Architecture Pattern.

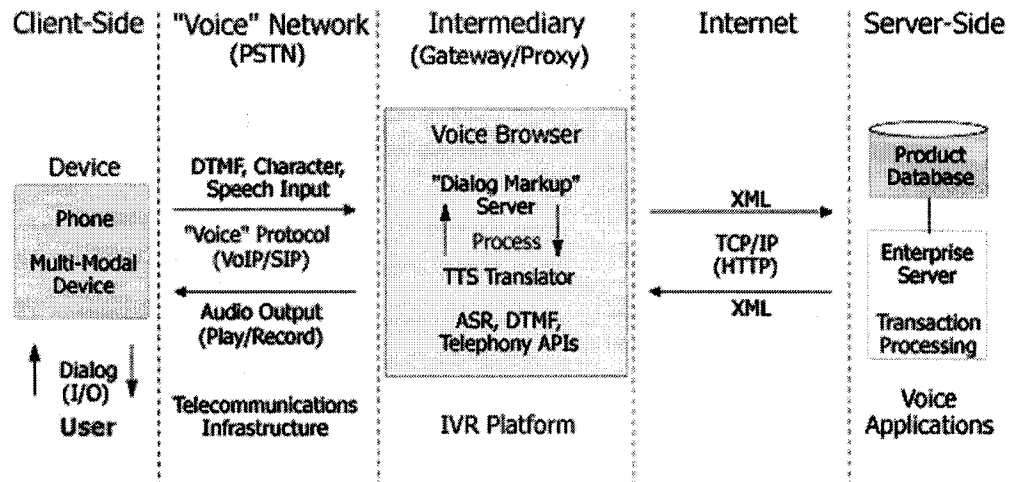


Figure 2.8. Mobile System with Speech Interface Architecture Pattern.

2.4.1.3. Mobile Application Patterns

Mobile Application Patterns are the different types of mobile applications with a recurring theme. There are several components that constitute a mobile application, leading to further layers of patterns. We discuss them in detail in the next section.

2.5. Classification of Mobile Application Patterns

Engineering a mobile application requires a process leading to a final product (the application itself). The product is then ultimately used by a user. This phenomenon presents a recurring theme, leading to four classes of patterns: Mobile Application Process Patterns, Mobile Application Architecture Patterns, Mobile Application Product Patterns, and Mobile Application Usage Patterns.

2.5.1. Relationship Across Patterns

The business requirements will drive the Mobile Application Process, which will lead to a definition of a Mobile Application Architecture. The Mobile Application Architecture in turn will assess the necessary Mobile Application Products, which in turn will affect the Mobile Application Usage. Mobile Application Usage practices give rise

to use cases that can influence the Mobile Application Process. Figure 2.9 reflects this inter-relationship.

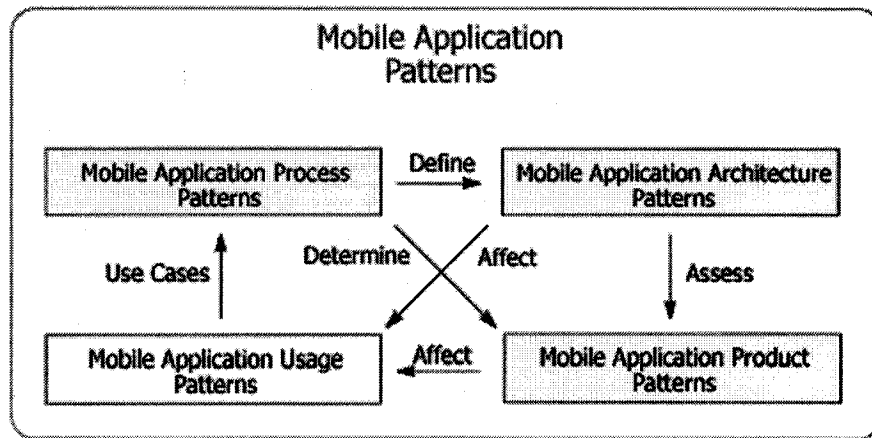


Figure 2.9. Classification of and Inter-Relationship among Mobile Application Patterns.

2.5.2. Mobile Application Process Patterns

The Mobile Application Process Patterns encapsulate the different processes followed to realize the mobile application. They involve a combination of managerial, strategic, and technical decisions. Identification of these is necessary, as they not only provide directions and constraints that lead to the completion, but also determine the quality of the mobile application. Although the Mobile Application Process will be driven by Business Patterns, a certain level of decoupling is necessary for insulation. This is because the impact of changes in the business requirements needs to be minimized and should not entirely altering the Mobile Application Process.

For our Mobile Application Process Patterns, we adopt the most widely tested and deployed Software Process Models: the Waterfall Model, the Evolutionary (Incremental Model), the Prototyping Model, and the Spiral Model. Each of these models has their own advantages and disadvantages. The suitability is determined by the nature of the application on a case-by-case basis.

2.5.3. Mobile Application Architecture Patterns

The Mobile Application Architecture Patterns represent the information architecture of a mobile application. It addresses how the mobile application is organized, what the resulting topology is, and how the different components are related to each other. Figure 2.10 shows four commonly used architecture patterns: Sequential, Hierarchical, Grid, and Star Architecture Patterns. Figure 2.11 shows how Sequential and Hierarchical Architecture Pattern can be combined to yield a new pattern.

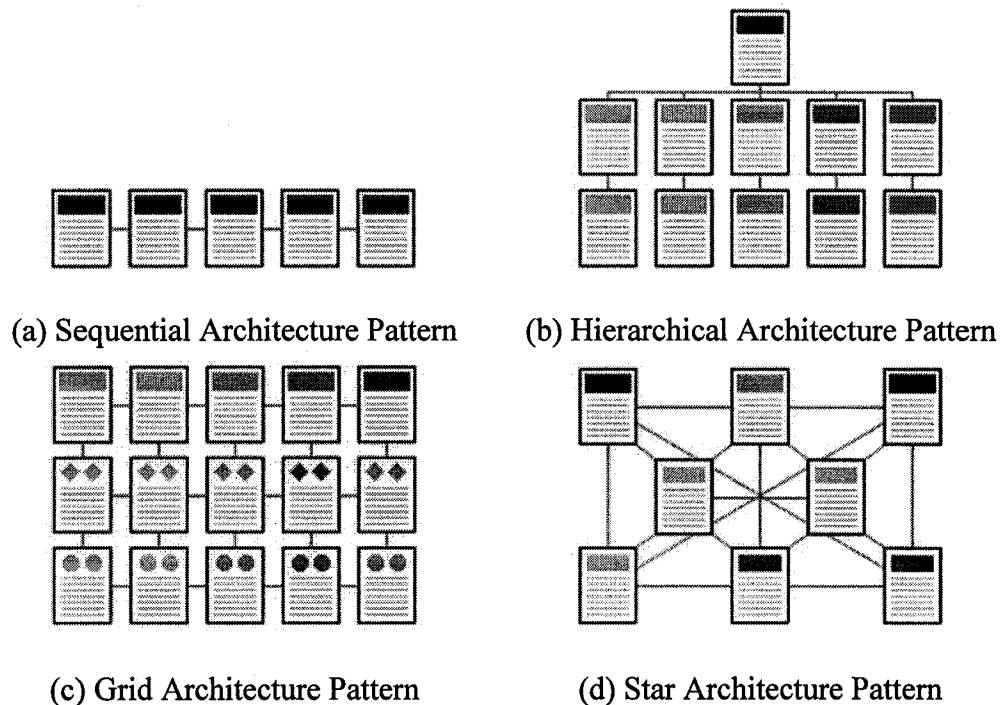


Figure 2.10. Mobile Application Architecture Patterns.

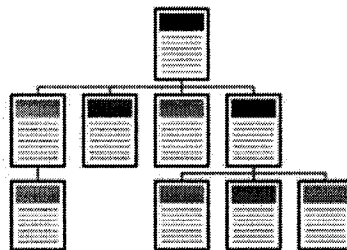


Figure 2.11. Combination (Sequential and Hierarchical) Architecture Pattern.

2.5.4. Mobile Application Product Patterns

The Mobile Application Product Patterns encapsulate the different products that engineers use or create to finally form the mobile application. Identification of these is necessary, as they shape the mobile application itself.

2.5.4.1. A Tiered Architecture of Mobile Application Product Patterns

Product Patterns can be classified further into patterns at three levels: Initial, Intermediate (or "Middle") and Final. Intermediate Product Patterns can be further divided into Autonomous (Independent or Standalone) and Non-Autonomous Patterns.

Figure 2.12 provides an illustration of the tiered classification of Product Patterns. There are four tiers (or layers) in the architecture to which, for traceability, we assign levels 0, 1 (1.1, 1.2) and 2. There is a uni-directional dependency among these tiers -- the outer tiers are dependent on the inner tiers. It is clear that the complexity increases as we move from the inside to the outside.

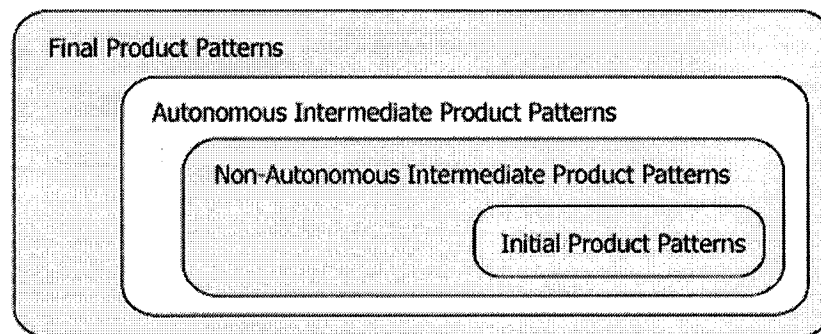


Figure 2.12. A Hierarchical Classification of Product Patterns in Mobile Applications. We now describe each tier in detail.

2.5.4.2. Initial Product Patterns

The Initial Product Patterns are the primitives that form the foundation on which we build applications. They are the atomic units that constitute a mobile application. Table 2.1 shows some Initial Product Patterns and example implementations based upon XML-related technologies.

| Pattern | | Example Implementation(s) |
|----------------------------------|--|---|
| Text Block Object | Paragraph | <ul style="list-style-type: none"> • XHTML Basic Structure Module |
| Table Object | | <ul style="list-style-type: none"> • XHTML Basic Table Module |
| List Object | | <ul style="list-style-type: none"> • XHTML Basic Structure Module • XHTML Basic Menu |
| Menu Object | | <ul style="list-style-type: none"> • VoiceXML Menu • XHTML Basic Form |
| Form Object | Visual Interface Form | <ul style="list-style-type: none"> • WML Form • XHTML Basic Form • Mobile Web Forms |
| | Speech Interface Form | <ul style="list-style-type: none"> • VoiceXML Form |
| | Generic Form | <ul style="list-style-type: none"> • XForms Form |
| Link Object | Simple Link Model [Motivated by HTML] | <ul style="list-style-type: none"> • VoiceXML Link • WML Link • XHTML Basic Link |
| | Generic Link | <ul style="list-style-type: none"> • XLink |
| Namespace Object | | <ul style="list-style-type: none"> • XHTML Namespace |
| Resource Identifier Object | Addressing | <ul style="list-style-type: none"> • URI [URL URN] • IRI |
| Embedded Object Mechanism Object | | <ul style="list-style-type: none"> • XHTML Basic Object Module • XHTML Basic Image Module • VoiceXML Audio Element |
| Documentation Object | Comments | <ul style="list-style-type: none"> • XML Comments |

Table 2.1. Initial Product Patterns of XML-Based Mobile Markup Language Elements with Example Implementation(s).

REMARKS

- An important consideration for this thesis is that the Initial Patterns must be implemented in XML and related technologies. Non-XML data formats will only be used in cases where either an XML solution is not available or is not feasible.
- We could consider the universe of mobile applications as a vector space. This class of Initial Patterns generates (when aggregated in a semantically meaningful way) other elements of this vector space, that is, Intermediate and Final Patterns. (We could also view Initial Patterns as the "Basis" Patterns, inspired by the notion of the basis of a vector space.)

2.5.4.3. Intermediate Product Patterns

Initial Patterns can be combined to form Intermediate Product Patterns. This can be carried out in different ways, giving rise to (at least) two different types of Intermediate Product Patterns.

2.5.4.3.1 Non-Autonomous Intermediate Product Patterns

Non-Autonomous (Dependent) Intermediate Product Patterns are directly composed of the Initial Patterns. They do not function entirely on their own and require other patterns to be functional. Table 2.2 shows some Non-Autonomous Intermediate Product Patterns and example implementations based upon XML-related technologies.

| Pattern | | Example Implementation(s) |
|---|-------------------------------------|--|
| [Business-Specific/Arbitrary] XML Document Object | Visual-Interface Document Object | <ul style="list-style-type: none">• SMIL Basic Document• Mobile SVG Document<ul style="list-style-type: none">◦ SVG Basic Document◦ SVG Tiny Document• WML Document [WML Card]• XHTML Basic Document |
| | Speech-Interface Document Object | <ul style="list-style-type: none">• VoiceXML Document |
| Image Object | | <ul style="list-style-type: none">• JPEG File• PNG File |

| | | |
|---|-----------------------------|---|
| | | <ul style="list-style-type: none"> • WBMP File |
| Animation Object | | <ul style="list-style-type: none"> • Flash File • SMIL Basic Animation Module |
| Audio Object | | <ul style="list-style-type: none"> • MP3 File |
| Video Object | | <ul style="list-style-type: none"> • MPEG-4 File |
| User Agent Extension Mechanism Object | | <ul style="list-style-type: none"> • ActiveX Control • Java Applet • Plug-In |
| Style Sheet | | <ul style="list-style-type: none"> • CSS Style Sheet • XSL Style Sheet |
| Transformer | | <ul style="list-style-type: none"> • XPathScript Script • XSLT Style Sheet • XTAL Script |
| Transclusion (Dynamic Data Inclusion)/Server Extension Mechanism Object | | <ul style="list-style-type: none"> • ASP • CGI • Enterprise Java Beans (EJBs) • Java Servlet • JSP Document • JSP Page • PHP |
| Client-Side Script Object | | <ul style="list-style-type: none"> • ECMAScript [JavaScript, JScript] • VBScript • WMLScript |
| Grammar Object | Document Definition Grammar | <ul style="list-style-type: none"> • XML DTD • XML Schema |
| | Speech Grammar | <ul style="list-style-type: none"> • JSGF • W3C XML Grammar • W3C ABNF Grammar |
| Annotation Object | Metadata Formats | <ul style="list-style-type: none"> • DCMES • RDF • SVG Metadata |

| | | |
|--|--|---|
| | | <ul style="list-style-type: none"> • XHTML Metadata Module |
|--|--|---|

Table 2.2. Non-Autonomous Intermediate Product Patterns with Example Implementation(s).

2.5.4.3.2 Autonomous Intermediate Product Patterns

Autonomous (Independent or Standalone) Intermediate Product Patterns are composed of the Non-Autonomous Intermediate Patterns. They are functional on their own. Table 2.3 shows an instance of an Autonomous Intermediate Product Pattern and example implementations based upon XML-related technologies.

| Pattern | | Example Implementation(s) |
|-------------------|--------------------------|---|
| Navigation System | Visual-Navigation System | <ul style="list-style-type: none"> • Mobile SVG <ul style="list-style-type: none"> ◦ SVG Basic ◦ SVG Tiny • WML • XHTML Basic |
| | Speech-Navigation System | <ul style="list-style-type: none"> • VoiceXML |

Table 2.3. Autonomous Intermediate Product Patterns with Example Implementation(s).

2.5.4.4. Final Product Patterns

Final Product Patterns are composed of the Autonomous Intermediate Patterns. They are the "last stage" in the mobile application development, and provide a complete solution to a mobile business problem. That is, a Final Product Pattern is a functional mobile application itself. Examples of Final Product Patterns are mobile location-based services, map servers, transactional services (such as mobile payments and mobile banking), mobile shopping carts, public information services (such as emergency 911 response systems, and travel and tourism information systems), mobile office, and mobile portals.

2.5.4.5. Example. Classification of a Mobile Commerce Application into Components

This example illustrates how a given mobile shopping portal application can be decomposed into sub-parts up to a semantically-related and meaningful level of graininess to yield different pattern levels.

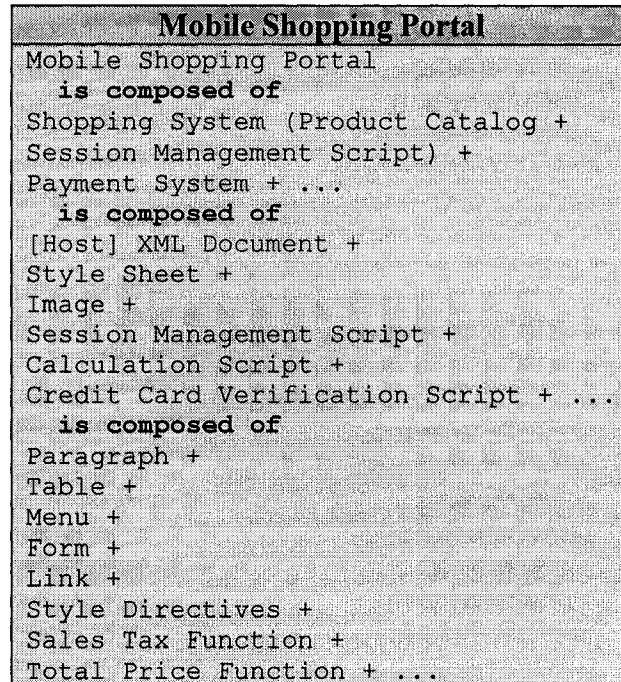


Figure 2.13 shows the abstract relationship between the example scenario discussed above and our pattern classification.

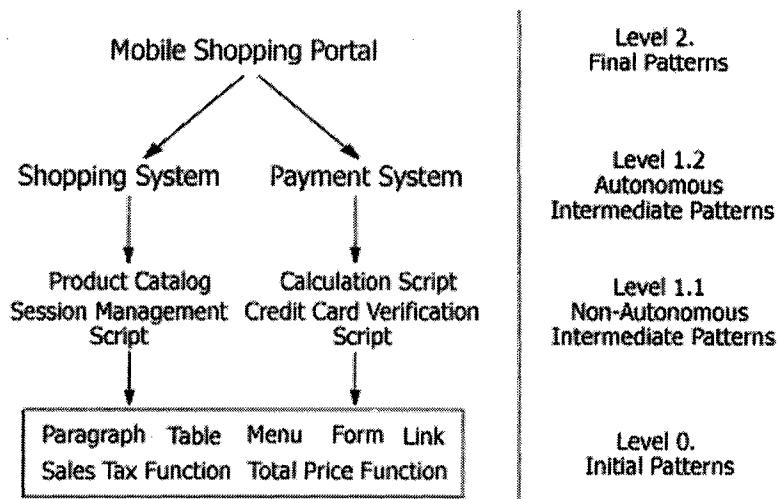


Figure 2.13. A Mobile Commerce Application Scenario with Pattern Levels.

2.5.4.6. Relationship to XML Vocabulary Design Pattern Collections

There are several independent collections of patterns [5, 6, 17, 36] for designing XML-based vocabularies. XML Accessibility Guidelines [62] also provide some suggestions towards designing vocabularies that could be encapsulated and formalized into patterns. The markup language element patterns (such as, Form, Link and so on), a subset of our Initial Product Patterns, are directly related to these XML Structural Design Patterns.

The languages that have taken the above patterns into consideration are likely to have a better design than those that have not. This can play a significant role in making a suitable choice when deciding upon a markup language for mobile applications. For example, document instances of languages that are based upon Separate Metadata and Data Pattern (say in a Head-Body Pattern and Metadata First Pattern) [36], are likely to be more human-readable and more comprehensible.

2.5.4.7. Relationship to J2EE Patterns

J2EE Patterns [3] is a collection of J2EE-based solutions to common problems, reflect the collective expertise and experience of Java technology architects at the Sun

Microsystems Java Center. A tiered approach is used to organize the J2EE patterns according to functionality. The resulting pattern catalog consists of five categories.

1. **Client Tier** patterns represent all device or system clients accessing the system or the application. A client tier pattern can, for example, be a WAP phone, or some device.
2. **Presentation Tier** patterns encapsulate all presentation logic required to service the clients that access the system. The presentation tier intercepts the client requests, provides single sign-on, session management and accesses business services, constructs the response, and delivers the response to the client. The presentation tier patterns can, for example, be servlets and JSPs.
3. **Business Tier** patterns provide the business services required by the application clients. The tier contains the business data and business logic. All business processing for the application is centralized into this tier. The business tier patterns can, for example, be EJBs.
4. **Integration Tier** patterns are responsible for communicating with external resources and systems, such as data stores and legacy applications. The business tier plays the role of middleware -- it is coupled with the integration tier whenever the business objects require data or services that reside in the resource tier. The integration tier patterns can, for example, be patterns related to JMS and JDBC.
5. **Resource Tier** patterns contain the business data and external resources. The resource tier patterns can, for example, be patterns related to legacy systems and business-to-business (B2B) systems, and services such as credit card authorization.

The Product Patterns that we have identified are related to J2EE Patterns collection. For example, Java implementations of our Transclusion Patterns in the Intermediate Product Patterns category are related to the J2EE Presentation Tier patterns. J2EE Patterns are limited to Java based enterprise solutions.

2.5.4.8. Relationship to Web Application Design Patterns

Hypermedia Design Patterns Repository [68] includes patterns for use in hypermedia design. They provide patterns for Interface/Layout, Structure/Navigation, and Content Design in hypermedia and Web applications. The collection is primarily limited to visual interfaces inspired by HTML, and does not include any implementation details. Our Initial, Autonomous, and Non-Autonomous Mobile Product Patterns that can give rise to mobile visual interfaces (such as, XML Visual-Interface Document, Menu, Form, Animation File, Video File, and so on) are related (at least in principle) to Hypermedia Design Patterns.

2.5.5. Mobile Application Usage Patterns

Mobile Application Usage Patterns encapsulate the different scenarios that the user comes across when interacting with the mobile application. Identification of these is the critical first step towards formalizing use cases and resolving usability issues. Examples include "Navigate", "Login/Logout" and "Fill-a-Form".

2.5.5.1. Relationship to Web Usability Patterns

User interfaces for the Web give rise to Usability Patterns. Several such patterns have been identified [39] in the UPAD Framework with a focus on "ease-of-use". The collection is primarily limited to visual interfaces inspired by HTML, and does not include any implementation details.

Our Initial, Autonomous, and Non-Autonomous Mobile Product Patterns that can give rise to mobile visual interfaces (such as, XML Visual-Interface Document, Menu, Form, Animation File, Video File, and so on) are related (at least in principle) to these Web Usability Patterns.

2.5.6. Relationship to General Pattern Taxonomies

Since the active use of patterns in Computer Science and related fields in the past decade, a reasonable consensus seems to have been reached on taxonomy of patterns. The

ones relevant in our case are the following:

1. **Architectural Patterns.** These deal with the highest levels of the system under design and capture recurring solutions to architectural problems. Our class of Mobile Application Process Patterns and Mobile Application Product Patterns overlaps with this category.
2. **Design Patterns.** These capture recurring solutions to the creation, structure, or behavioral characteristics of objects in object-oriented systems. Our class of Mobile Application Product Patterns and Mobile Application Usage Patterns overlaps with this category.
3. **Idioms.** These represent recurring solutions to the problems faced by writers of a specific formal (markup, programming) language. Our sub-class of Mobile Application Initial Product Patterns overlaps with this category.

2.6. Pattern Language

A pattern language [4] formalizes the definition of a pattern by providing a vocabulary and helps a user in understanding that pattern. It also provides a notation for writing future patterns, thereby extending the existing pattern catalog in a uniform fashion.

Some of the defining characteristics of a pattern language inspired by software engineering principles are: abstraction, anticipation of change, generality, semantic-relationship (among patterns) that may lead to inheritance (of the solution of one pattern by the other, in whole or in part), and consistency (that one pattern should not provide a solution that conflicts with that of the other).

2.6.1. Limitations of Traditional Pattern Language Notation

The "traditional" pattern language notation that has been often used looks like the following:

```
IF you find yourself in CONTEXT
  For example EXAMPLES,
  With PROBLEM,
  Entailing FORCES
THEN for some REASONS,
  Apply DESIGN FORM AND/OR RULE
  To construct SOLUTION
  Leading to NEW CONTEXT and OTHER PATTERNS
```

This form has been extended in some cases to accommodate patterns for specific areas [16, 39]. However, this notation (and its variations) has several limitations:

- The traditional pattern language notation makes broad use of natural language-based prose and has little context. Hence, the patterns based on them are strongly "document-oriented", and can seem vague and open to interpretation. This limits their use in automated processing environments and interchange.
- The notation is paper-printing oriented and does not provide any means of using the potential and benefits offered by the electronic medium of communication.

This has several implications:

- There is no standard way in which non-text objects related to pattern (say, solution or scenario-of-use) can be included.
- There is no standard way in which instances of pattern expressed in the notation can be interchanged across a broad range of devices.
- There is no explicit support for presenting patterns expressed in the traditional notation on Internet-based information systems, such as the Web. (The patterns could be written using some word processing package that can associate presentation and linking semantics, and the result can be converted to, for example, HTML. However, such process is not automatic and resulting documents are highly inefficient.)
- There is no standard way to verify whether a pattern is compliant with the notation.
- There is no standard way in which pattern solution implementations can be included.
- There is no explicit support for Internationalization, in particular, for non-English language characters or other special symbols.

These limitations have motivated us to take an alternative approach to devise a pattern notation that can resolve these issues as well as to provide other features that can adequately represent mobile application patterns. Expressing mobile patterns in XML bypasses above issues and provides various advantages.

2.6.2. The Significance of XML Representation of Patterns and the Motivation for MAPML

Apart from the benefits of XML mentioned in Section 2.2, there are several advantages of expressing mobile patterns in XML:

1. XML provides a standardized way patterns are written, and hence facilitates the share and re-use of patterns.
2. XML is more expressive and provides a higher level of granularity than that is possible by pure natural language use.
3. XML provides a way to more formally specify patterns.
4. XML allows a pattern service provider (a pattern server) to vary pattern presentations to suit different situations, such as, different client-side requirements.
5. XML makes it feasible to verify (validate against a grammar) the pattern format.
6. XML enables cross-referencing of patterns using hyperlinking mechanisms.
7. XML enables automated programs to extract summaries or descriptions of patterns for the purpose of indexing and use in pattern catalogs.
8. XML provides structure, which facilitates contributors in submitting patterns via the Web or otherwise. Making pattern catalogs available on the Web provides various advantages: evolution (for maintenance, extension), global instant access, inclusion of "dynamic" objects as part of solution, and so on.
9. XML makes it easier for users to (via links) navigate or (via forms) search the pattern collection.
10. XML enables Pattern Catalogs to be used as a knowledge base.

Inspired by above possibilities, we propose MAPML – an XML-based notation to represent patterns in MAPCLASS. Note that a pattern expressed in MAPML can always

be down-transformed to a traditional text-based notation, such as the one mentioned in Section 2.6.

2.6.3. MAPML and XML

MAPML, as a markup language based on XML, inherits many of its properties. This section summarizes the key notions necessary for the thesis.

Each MAPML document consists of a sequence of characters, which may represent markup or character data (non-markup). Each character belongs to the ISO/IEC 10646 [28] character set. ISO/IEC 10646 is character-by-character equivalent to Unicode [45], which is a standard, interoperable way to implement ISO/IEC 10646. When MAPML documents are sent by servers as a stream of bytes, user agents interpret them as a sequence of characters. User agents must know the underlying character encoding to correctly interpret MAPML documents. Character encoding is a method of converting a sequence of bytes into a sequence of characters, and vice versa. Commonly used character encoding that supports most Western European languages on the Web are ISO-8859-1 ("Latin-1") and UTF-8 (encoding of ISO 10646 using a different number of bytes for different characters).

Markup provides logical structure to a MAPML document and is responsible to represent the element and attribute types of MAPML. MAPML element types (or just elements) represent capabilities (structure or desired behavior) of the mobile patterns described in Section 2.4. MAPML attribute types (or just attributes) provide properties associated with these elements.

2.7. Summary

Mobile applications provide various benefits to businesses and to the consumer. XML can play a key role in constructing these mobile applications. The classification of mobile application patterns presented in this chapter helps us to visualize the macroscopic and microscopic structure of a pattern. Patterns must have a notation that is interoperable, interchangeable, and presentable across different computing environments. XML provides such a notation.

Chapter 3

MAPML Requirements and Design

For both longevity and traceability, it is necessary that MAPML follow a planned approach. This chapter provides an insight into the process of creating MAPML. Section 3.1 lists the use cases that present instances of how MAPML could be used. Section 3.2 describes the functional and non-functional requirements that outline the goals and scope of MAPML. Finally, Section 3.3 discusses the architectural and detailed design considerations based on the constraints set by the requirements.

Software engineering principles that have guided the MAPML requirements and design are separation of concerns, generality, anticipation of change, and rigor.

To indicate requirement or design levels, a prioritization scheme is desirable. The keywords “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “RECOMMENDED”, “MAY”, and “OPTIONAL” used in Sections 3.2 and 3.3 are to be interpreted as described in IETF RFC 2119 [9]. For the sake of consistency and flow with the rest of the text, they may not appear in uppercase.

3.1. MAPML Use Cases

A **use case** [30] describes the sequence of interactions between actors and the system necessary to deliver the service that satisfies a certain goal. It captures who (actor) does what (interaction) with the system, for what purpose (goal), in an implementation-independent and system-independent manner.

Use cases have been widely used for primarily functional requirements elicitation. Table 3.1 provides a summary of some typical MAPML use cases that have been identified. The first column of the table indicates the actors. The three major actors are the pattern author, the pattern user, and the pattern (catalog) administrator. The next two columns of the table indicate the corresponding interactions and goals, respectively. The

last column of the table includes the labels of requirements (from Section 3.2) that these use cases have helped define.

| Actor | Interaction | Goal | Associated Requirement(s) |
|-----------------------|--|--|--|
| Pattern Author | Include an SVG Basic Fragment in a MAPML Document. | To Provide a Graphic Solution to the Pattern Problem. | [MAPML-Requirement-Extensibility] |
| | Transform a MAPML Document to Plain Text. | To Read the Pattern Description on a TTY Terminal. | [MAPML-Requirement-Interoperability], [MAPML-Requirement-Presentability], [MAPML-Requirement-Transformability] |
| Pattern User | Search Pattern Catalog by Pattern Name or by Pattern Author. | To Find a Related Pattern. | [MAPML-Requirement-Identifiability-1], [MAPML-Requirement-Identifiability-2] |
| | Navigate through a Mobile Application Pattern Collection. | To View All Sub-Patterns of a Final Product Pattern. | [MAPML-Requirement-Genericity-2], [MAPML-Requirement-Navigability] |
| Pattern Administrator | Extract a List of Pattern Authors. | To Index. | [MAPML-Requirement-Processability-1], [MAPML-Requirement-Processability-2] |
| | Validate a MAPML Document. | To Check if the Pattern Format is Correct before Adding it to the Pattern Catalog. | [MAPML-Requirement-QA] |

Table 3.1. Examples of MAPML Use Cases.

The Unified Modeling Language (UML) [42] is a general-purpose object-oriented modeling language for specifying, visualizing, constructing, and documenting the artifacts of systems. UML has explicit support for use cases. Figure 3.1 illustrates a UML Use Case diagram for MAPML Transformation use case described in Table 3.1.

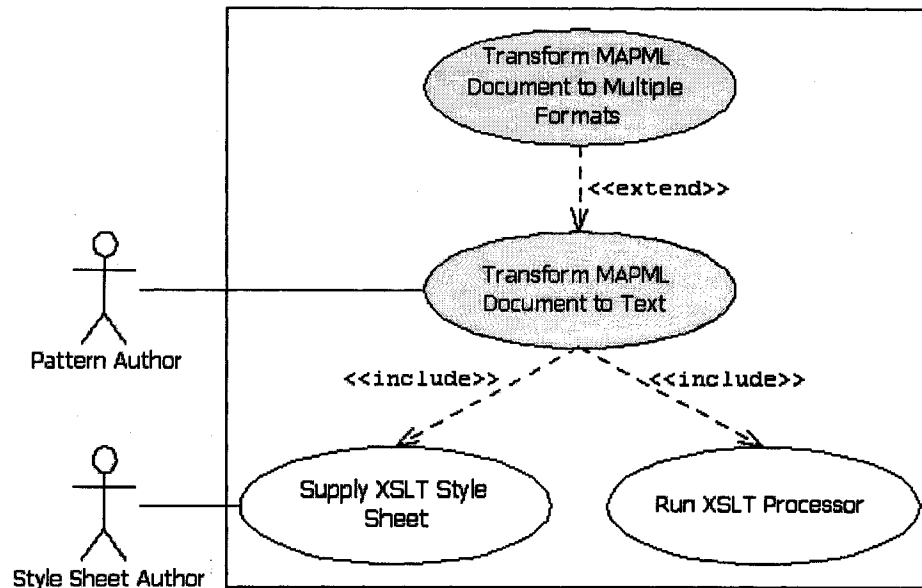


Figure 3.1. MAPML Transformation Use Case Diagram.

Scenarios are use case instances. Figure 3.2 shows a UML Sequence Diagram for MAPML Transformation use case scenario. A MAPML document goes through a sequence of events and is transformed to a text format. A rejection reflects a failure scenario that can occur if, for example, the input MAPML document or the style sheet is invalid.

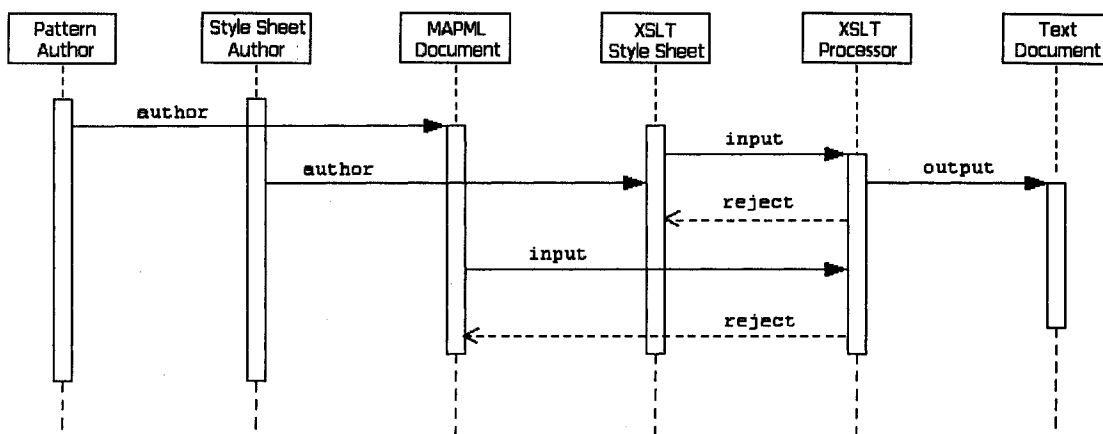


Figure 3.2. MAPML Transformation Use Case Scenario.

3.2. MAPML Requirements

This section outlines MAPML requirements in the categories of Association, Demonstrability, Documentation, Evolvability, Extensibility, Genericity, Identifiability, Interoperability, Legality, Minimality, Modularity, Navigability, Presentability, Processability, Quality Assurance, Re-Usability, Transformability, Uniqueness, and Usability. These categories are inspired by the IEEE 830 Recommended Practice for Software Requirements Specifications [26]. Each requirement is equipped with an identification label for traceability, and includes a rationale.

MAPML Requirements for Association

- **[MAPML-Requirement-Association]** MAPML must be able to show relationships among patterns.

Rationale. Patterns have interrelationships within and across pattern catalogs.

MAPML Requirements for Demonstrability

- **[MAPML-Requirement-Demonstrability]** MAPML must be able to demonstrate pattern solutions.

Rationale. A pattern needs to demonstrate that the solution it suggests actually works in the “real world”.

MAPML Requirements for Documentation

- **[MAPML-Requirement-Documentation]** MAPML must be well documented.

Rationale. For pattern problems and solutions expressed by MAPML to be understandable and for MAPML evolution to be traceable, components of MAPML need to be well documented. (See also, [MAPML-Requirement-Evolvability-1], [MAPML-Requirement-Evolvability-2], [MAPML-Processability-2], and [MAPML-Requirement-Usability-1].)

MAPML Requirements for Evolvability

- **[MAPML-Requirement-Evolvability-1]** MAPML shall be evolvable.

Rationale. Patterns and their taxonomies are not static. Practices that are considered “experimental” could lead to patterns in the future that may require changes in the definition of MAPML. Therefore, MAPML must be prepared to evolve.

- **[MAPML-Requirement-Evolvability-2]** MAPML documents shall be manageable. Specifically, MAPML documents must be self-sufficient for basic configuration management.

Rationale. If the pattern content changes (for example, changes in author address or ownership), MAPML document(s) will need to be adjusted accordingly. Hence, MAPML documents should not depend on external means for basic configuration so that the modification process can be autonomous. For sophisticated management, one can resort to tools such as RCS/CVS.

MAPML Requirements for Extensibility

- **[MAPML-Requirement-Extensibility]** MAPML documents shall be extensible. In particular, MAPML should provide the ability to include fragments of non-MAPML XML-based vocabularies in MAPML documents.

Rationale. Pattern solutions may need to make use of markup languages as part of the implementation. Markup based on these languages (that is not native to MAPML) will then need to be included in MAPML documents.

MAPML Requirements for Genericity

- **[MAPML-Requirement-Genericity-1]** MAPML shall be generic enough to be able to express all the aspects of the definition and all the components of a pattern.

Rationale. The fundamental objective of MAPML is to be able to completely represent the components of a pattern. According to the Patterns-Discussion FAQ [37], “a formal notation is not a pattern if it omits descriptions of context, the problem(s) it solves, evidence for adequacy of the solution, construction or implementation guidelines, or relations with other patterns.”

- **[MAPML-Requirement-Genericity-2]** MAPML shall be generic enough to be able to express all the classes of mobile patterns.

Rationale. MAPML needs to represent all patterns in MAPCLASS. Furthermore, it should also be prepared for any additions to it.

MAPML Requirements for Identifiability

- **[MAPML-Requirement-Identifiability-1]** MAPML must provide support for pattern name and its identifier.

Rationale. There must be a way to uniquely identify a pattern entry in a catalog.

- **[MAPML-Requirement-Identifiability-2]** MAPML must provide support for metadata related to the pattern and its author.

Rationale. Both the pattern catalog administrator and the pattern user need to know the details about the pattern, its author, and status to make further decisions.

MAPML Requirements for Interoperability

- **[MAPML-Requirement-Interoperability]** MAPML shall be interoperable. In particular, MAPML shall be useful with other data formats desirable for mobile patterns, be platform-independent, be vendor-neutral, and be device independent.

Rationale. MAPML documents need to function in unison with a variety of language environments, not be restricted by any proprietary hardware/network/software technology, and interoperate across a broad range of devices.

MAPML Requirements for Legality

- **[MAPML-Requirement-Legality]** MAPML shall support the legal rights of the pattern author.

Rationale. With the evolution of electronic media of information distribution such as the Web, the intellectual property rights of a pattern author are challenged, and should be protected at all times.

MAPML Requirements for Minimality

- **[MAPML-Requirement-Minimality]** MAPML shall be minimal.

Rationale. Although the traditional notation has several limitations, it is simple. MAPML needs to be simple and comprehensible so as to ease the transition.

MAPML Requirements for Modularity

- **[MAPML-Requirement-Modularity]** MAPML structure shall be modular.

Rationale. Modular structures significantly improve the evolvability, including maintainability and extensibility.

MAPML Requirements for Navigability

- **[MAPML-Requirement-Navigability]** MAPML documents shall be navigable within themselves and across other documents.

Rationale. MAPML document collections need to be navigable so that they can be cross-referenced, and related patterns can be discovered.

MAPML Requirements for Presentability

- **[MAPML-Requirement-Presentability]** MAPML shall be presentable on a broad range of devices. However, this should not impede the development of MAPML.

Rationale. MAPML documents need to be presentable in situations with “lowest common denominator” (monitors with limited capability for colors and resolution, limited font support, and so on), particularly on the Web. Still, following the Separation of Quality principle, the primary focus of MAPML is to interchange patterns. Presentation of patterns is secondary.

MAPML Requirements for Processability

- **[MAPML-Requirement-Processability-1]** MAPML should separate information internal to a pattern from information external to the pattern. Specifically, any pattern meta-information or information related to pattern presentation should not be part of pattern core.

Rationale. Decoupling pattern core from related information eases pattern processing and management.

- **[MAPML-Processability-2]** MAPML shall have coarse-graininess to be equally machine-processable ("data-oriented") and human-readable ("document-oriented").

Rationale. MAPML-based markup needs to be read by humans and processed by machines.

MAPML Requirements for Quality Assurance

- **[MAPML-Requirement-QA]** MAPML shall provide formal grammars with respect to which MAPML documents can be validated.

Rationale. Quality assurance of MAPML documents is critical. Existence of formal grammars makes it possible to test MAPML documents for conformance when required.

MAPML Requirements for Re-Usability

- **[MAPML-Requirement-Re-Usability]** MAPML shall make optimal use of the existing XML pattern taxonomies.

Rationale. To be well-defined, robust, and manageable, MAPML needs to consider existing “best practices” in XML.

MAPML Requirements for Transformability

- **[MAPML-Requirement-Transformability]** MAPML shall follow a format that can be easily transformed to other XML languages, particularly with “lower-structure”. However, this shall not impede the development of MAPML.

Rationale. MAPML documents need to be transformable to formats that can be read by a user with minimal client-side requirements (such as a plain-text editor).

MAPML Requirements for Uniqueness

- **[MAPML-Requirement-Uniqueness]** MAPML syntax must be universally unique.

Rationale. There always exists the potential that the syntactical constructs of two or more markup languages are in conflict. Furthermore, MAPML documents may need to be heterogeneous. Therefore, MAPML syntax needs to be uniquely identifiable among the presence of non-MAPML objects in MAPML documents.

MAPML Requirements for Usability

- **[MAPML-Requirement-Usability-1]** MAPML syntax shall be comprehensible to the pattern community.

Rationale. MAPML needs to leverage on a vocabulary that is widely-used and understood by existing base of pattern users.

- **[MAPML-Requirement-Usability-2]** MAPML shall be expressed in a format that supports accessibility.

Rationale. MAPML documents need to have as broad user-base as possible, particularly since it expresses patterns for both visual and speech interfaces.

- **[MAPML-Requirement-Usability-3]** MAPML shall provide support for internationalization.

Rationale. MAPML documents needs to have as broad user-base as possible, particularly since the pattern users are spread all around the world.

3.3. MAPML Design Description

This section outlines the MAPML design descriptions corresponding to the categories of Section 3.2. Some of the design decisions have been inspired by the IEEE STD 1016-1987 Recommended Practice for Software Design Descriptions [25]. Each design description is equipped with an identification label for traceability.

MAPML Design Description for Association

- **[MAPML-Design-Association]** MAPML must provide an element name `pattern.related` that provides the names, type of relationship, and links to related patterns.

MAPML Design Description for Demonstrability

- **[MAPML-Design-Demonstrability]** MAPML must provide an element name `scenario` that can include examples in different forms (narrative text, figure, code, markup) to demonstrate evidence of pattern solution in use.

MAPML Design Description for Documentation

- **[MAPML-Design-Documentation]** MAPML must provide elements for figure caption, pattern terminology, and for including arbitrary text.

MAPML Design Description for Evolvability

- **[MAPML-Design-Evolvability-1]** MAPML must allow changes to its modules as well as addition of other modules to it.
- **[MAPML-Design-Evolvability-2]** MAPML must provide an attribute name `version` to track changes in MAPML Specification and MAPML documents.

MAPML Design Description for Extensibility

- **[MAPML-Design-Extensibility]** MAPML can be extended both at the grammar-level and at the document-level. A modular approach to MAPML (see [MAPML-Design-Modularity]) will ease the future extensions to MAPML grammars. MAPML documents can be extended to include fragments of non-MAPML XML-based vocabularies once MAPML elements and attributes have been uniquely identified (see [MAPML-Design-Uniqueness]).

MAPML Design Description for Genericity

- **[MAPML-Design-Genericity-1]** MAPML must provide elements that can encapsulate all aspects of a pattern: problem, context, constraint(s), solution, the impact of the solution, as completely as possible. These elements should be the children of the element name `pattern`.
- **[MAPML-Design-Genericity-2]** MAPML must provide an attribute name `class`, possible values of which are the different classes of mobile patterns identified in Chapter 1, Section 1.5.

MAPML Design Description for Identifiability

- **[MAPML-Design-Identifiability-1]** MAPML must provide an element for pattern name and an attribute name `id` that associates a unique identifier with each pattern.
- **[MAPML-Design-Identifiability-2]** MAPML must provide an element for encapsulating pattern metadata. This element should allow elements for pattern related information and pattern author related information to be its children.

MAPML Design Description for Interoperability

- **[MAPML-Design-Interoperability]** MAPML must follow the Device Independence Principles [61] and should not allow any device-specific information in its syntax.

MAPML Design Description for Legality

- **[MAPML-Design-Legality]** MAPML must provide an element which allows inclusion of copyright information regarding the pattern owner, type of pattern license, and terms-of-use of the pattern.

MAPML Design Description for Minimality

- **[MAPML-Design-Minimality]** MAPML should avoid verbosity by re-using established modules. MAPML should allow the elements and attributes that are not part of pattern core to be optional. This could be established by what is and what is not required in a pattern definition.

MAPML Design Description for Modularity

- **[MAPML-Design-Modularity]** MAPML should be structured into semantically-related modules. At the grammar-level, these modules should be implemented in XML Schema Languages (see Figure 3.3) that provide extensibility, namely XML Schema and RELAX NG.

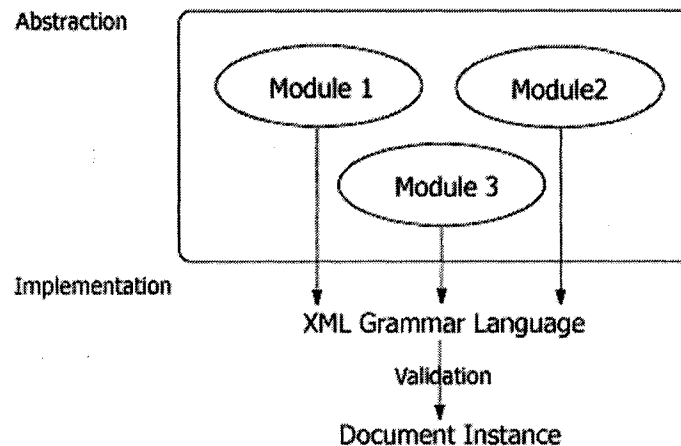


Figure 3.3. MAPML Modularization Process.

MAPML Design Description for Navigability

- **[MAPML-Requirement-Navigability]** MAPML must provide an element for linking resources that can be identified via a URI. MAPML should also adhere to a structure that is close to that of a tree.

MAPML Design Description for Presentability

- **[MAPML-Design-Presentability]** HTML [54] (along with its variants in the XHTML Framework [58]) is the predominant language on the Web for presentation, and therefore provides a viable option as a format to which MAPML documents can be transformed to. Therefore, MAPML should follow a format so that the documents based on it are readily transformable to XHTML Basic [57] and XHTML 1.1 [63]. MAPML must not provide any presentational elements or attributes. Presentation semantics should be relegated to style sheet languages.

MAPML Design Description for Processability

- **[MAPML-Design-Processability-1]** MAPML must have two element names: one for information internal to a pattern, and the other for information external to a pattern. These elements should be isolated from each other, that is, none is a child of the other. Specifically, MAPML could have a head and a body element names that allow inclusion of pattern metadata in the former and pattern content in the latter.
- **[MAPML-Processability-2]** MAPML should allow use of natural language text in its elements wherever necessary. It should provide elements for pattern metadata, abstract, and description.

MAPML Design Description for Quality Assurance

- **[MAPML-Design-QA]** MAPML grammars should be as expressible as possible. MAPML must provide formal grammars based on XML Schema (normative), EBNF (informative), XML DTD (informative), and RELAX-NG (informative). These grammar themselves must also be validated.

MAPML Design Description for Re-Usability

- **[MAPML-Design-Re-Usability]** MAPML should refer to patterns in taxonomies for XML structural design, such as, in XML Characterization Project [32] and the XML Patterns Catalog [36].

MAPML Design Description for Transformability

- **[MAPML-Design-Transformability]** MAPML should follow a format so that a transformation code, script, or style sheet can easily observe the structure of an arbitrarily complex document instance. Mixed content models should be avoided as far as possible.

MAPML Design Description for Uniqueness

- **[MAPML-Design-Uniqueness]** MAPML must support Namespaces in XML [49], providing both a URI and a prefix. MAPML namespace can then be used to uniquely identify MAPML elements and attributes in MAPML documents that contain fragments of non-MAPML XML-based vocabularies.

MAPML Design Description for Usability

- **[MAPML-Design-Usability-1]** MAPML should use pattern terminology in its element names as much as possible. Any extensions should be clearly justified. To help a pattern user, MAPML should provide elements that can express the rationale behind the pattern, the pattern solution structure, the pattern solution strategy, example implementation of the pattern solution, and the scenario where the pattern has been used.
- **[MAPML-Design-Usability-2]** MAPML should conform to Web Content Accessibility Guidelines 1.0 [50] in general, and XML Accessibility Guidelines [62] in particular. These guidelines provide in-depth details of how, in general, documents based on markup languages can be made accessible to an audience that is geographically-dispersed and/or with physiological disabilities. In particular, MAPML should provide an alternate means to describe the information contained in non-textual objects (such as, images).

- **[MAPML-Design-Usability-3]** MAPML should make use of the global attribute provided by XML (`xml:lang`) to support Internationalization. Furthermore, for documents that claim conformance to MAPML, inclusion of the character encoding attribute in the XML processing instruction should be made a requirement. That is, all MAPML documents should begin with

```
<?xml version="1.0" encoding="..."?>
```

where appropriate value of the character encoding is included. For example, a MAPML document containing only English language characters can begin with

```
<?xml version="1.0" encoding="UTF-8"?>
```

3.4. Summary

In this chapter, we described the ideological, social, and technical decisions behind the definition of MAPML. The objectives and constraints set in the requirements and approaches in the design lead to the implementation of MAPML Specification (Chapter 4) and MAPML utilities (Chapter 5).

Chapter 4

MAPML Specification

This chapter completely defines the syntax and semantics of the MAPML 1.0 Specification. Section 4.1 lists the MAPML modules and the elements they consist of. Section 4.2 lists the basic properties of elements and attributes, including conditions on their data types and enumeration. Sections 4.3 to 4.7 provide definitions of individual elements along with the details of corresponding attributes, sub-elements, and examples. Section 4.8 provides definitions of each of the attributes. Section 4.9 provides objects of MAPML identification, namely MAPML media type and MAPML namespace. Finally, Section 4.10 discusses the issue of conformance with respect to MAPML.

The current status, unless stated otherwise, of all MAPML related documents, grammars, and software is Version 1.0.

4.1. MAPML Modules and Element Definitions

MAPML is composed of five sets of semantically-related elements and attributes called **modules**: Association Module, Meta-information Module, Problem Module, Solution Module, and Structure Module.

MAPML has thirty elements, each responsible for a specific functionality of the language.

Table 4.1 lists the MAPML modules along with the alphabetical list of corresponding elements.

| Module | Elements |
|-------------------------|---|
| Association Module | class, link, pattern.related, reference |
| Meta-Information Module | abstract, author, caption, date, description, keyword, license, metadata, name, term, title |

| | |
|-------------------------|---|
| Problem Module | constraint, context, problem |
| Solution Module | consequence, implementation, object, rationale, scenario, solution, strategy, structure |
| Structure Module | body, head, mapml, pattern |

Table 4.1. MAPML Modules.

Figure 4.1 gives a structural overview of MAPML that display its key elements.

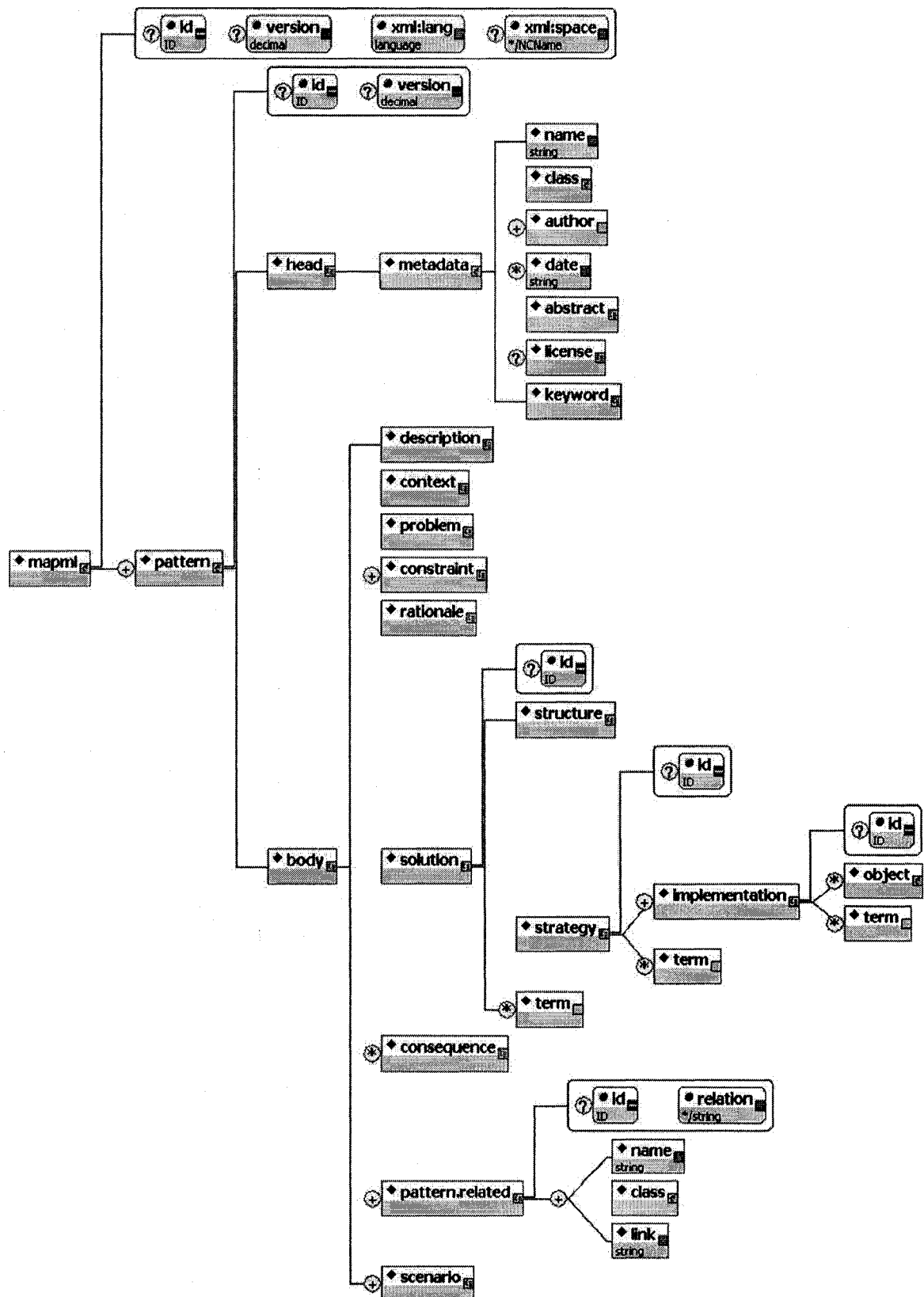


Figure 4.1. Structure of MAPML.

4.2. Properties of MAPML Elements and Attributes

The two properties of interest are data type and enumeration.

4.2.1. MAPML Data Types

MAPML data types determine the type of data allowed in element content and attribute values.

A "**Singleton**" data type indicates that the element content consists of only character data and does not have any sub-elements. An "**Aggregate**" data type indicates that the element content consists of only sub-elements. A "**Mixed**" data type indicates that the element content is a mixture of character data and sub-elements. The class of characters allowed as character data are determined by the XML 1.0 Specification (see Section 2.6.3). For the types of character data in element content and attribute values, a library of data types such as the one provided by XML Schema [59] can be used.

Empty elements do not contain any content. All MAPML elements, unless stated otherwise, are non-empty.

4.2.2. Enumeration

Enumeration determines the number of times an element can occur, that is, its multiplicity.

The occurrence of elements is classified as **Required**, **Conditional**, or **Optional**. An element that is labeled as "**Required**" must be present in every MAPML document; an element that is labeled as "**Conditional**" must be present in a MAPML document under the given conditions; an element that is labeled as "**Optional**" may not be present in every MAPML document. Related to occurrence is the concept of **cardinality**, which indicates the number of times an element can occur, if at all. If an element is a child of another element, the cardinality indicates the number of times the child element can occur in its parent. A cardinality of "**N**" indicates that the element occurs N times, where N=1, 2, ... A cardinality of "**N..M**" indicates that the element can occur N to M times, where N=0, 1, 2, ..., M=1, 2, ..., and M>=N. A cardinality of "**N..Unbounded**" indicates that the

element can occur N or more times, where N=0, 1, 2, ... An element whose occurrence is **Required** will have a cardinality of (at least) 1; an element whose occurrence is either **Optional** or **Conditional** will have a cardinality of (at least) 0. Note that the root element is always **Required** with cardinality = 1.

The occurrence of attributes is classified as **Required**, **Conditional**, or **Optional**, and is indicated in the parenthesis next to its name. An attribute that is labeled as "**Required**" must always be present in its corresponding element; an attribute that is labeled as "**Conditional**" must be present in its corresponding element under the given conditions; an attribute that is labeled as "**Optional**" may not always be present. A "**None**" indicates that the element does not have any attribute. An attribute can occur only once in an element.

The foregoing sections specify the individual definitions of elements and attributes.

4.3. Association Module

The Association Module signifies association of the pattern to its class, to other pattern(s), or to a reference. The association is made possible by the linking mechanism.

The class Element

| | |
|-------------------------------------|---|
| Element Name | class |
| Related MAPML Requirement(s) | [MAPML-Requirement-Association], [MAPML-Requirement-Genericity-2] |
| Module | Association Module |
| Description | Mobile application patterns are classified into several categories as discussed in Chapter 2, Sections 2.4 and 2.5. The <code>class</code> element names the classification scheme to which the pattern belongs to. |
| Data Type | Singleton |
| Attribute(s) | id (Optional) |
| Parent Element(s) | metadata |
| Child Element(s) | None |

| | |
|----------------|---|
| Example | <code><class>Initial Product Pattern</class></code> |
|----------------|---|

Table 4.2. The class Element

The link Element

| | |
|-------------------------------------|---|
| Element Name | link |
| Related MAPML Requirement(s) | [MAPML-Requirement-Navigability] |
| Module | Association Module |
| Description | <p>Patterns belonging to one catalog, like MAPCLASS, are closely interconnected. The notion of linking makes these interconnections possible and provides a functionality for local and global cross-referencing.</p> <p>The purpose of the link element is to provide linking semantics to the pattern elements it is associated with.</p> |
| Data Type | Not Applicable (Empty Element) |
| Attribute(s) | id (Optional), uri (Required) |
| Parent Element(s) | object, reference |
| Child Element(s) | None |
| Example | <code><link uri="http://www.mapml.com/images/ebook.svg"/></code> |

Table 4.3. The link Element

REMARKS

When transforming a MAPML document to other formats, the link element can be mapped to sophisticated hyperlinking schemes such as XLink [62], which provides support for both uni-directional and bi-directional linking.

The pattern.related Element

| | |
|-------------------------------------|----------------------------------|
| Element Name | pattern.related |
| Related MAPML Requirement(s) | [MAPML-Requirement-Genericity-1] |
| Module | Association Module |

| | |
|--------------------------|--|
| Description | The pattern.related element provides information on other pattern(s) that is related to the pattern being described. It may include relationship to the described pattern, class to which it belongs to, and pointer to where it can be found. |
| Data Type | Aggregate |
| Attribute(s) | id (Optional), relation (Required) |
| Parent Element(s) | body |
| Child Element(s) | name (Required, Cardinality = 1), class (Required, Cardinality = 1), link (Optional, Cardinality = 0..1) |
| Example | <pre><pattern.related relation="subordinate"> <name>MAPCLASS.Navigation</name> <class>Autonomous Intermediate Product Pattern</class> </pattern.related></pre> |

Table 4.4. The pattern.related Element

The reference Element

| | |
|-------------------------------------|--|
| Element Name | reference |
| Related MAPML Requirement(s) | [MAPML-Requirement-Documentation], [MAPML-Requirement-Usability-1] |
| Module | Association Module |
| Description | The reference element includes information on references related to the pattern problem and/or solution that may be helpful towards further understanding. It is suggested that a referenced item should be canonical and follows standard guidelines of a bibliography. If a referenced item is accessible via the Web, the URL should be provided. |
| Data Type | Aggregate |
| Attribute(s) | id (Required) |
| Parent Element(s) | problem, solution |
| Child Element(s) | title (Required, Cardinality = 1), author (Required, Cardinality = 1..Unbounded), date (Required, Cardinality = 1..Unbounded), link (Required, Cardinality = 1) |
| Example | <pre><reference id="appleton2000"> <title>Patterns and Software: Essential Concepts and</pre> |

| | |
|--|--|
| | <pre> Terminology</title> <author>Appleton, B.</author> <date>2000</date> <link uri="http://www.enteract.com/~bradapp/docs/ patterns-intro.html"/> </reference> </pre> |
|--|--|

Table 4.5. The reference Element

4.4. Meta-Information Module

Literate Programming [33] advocates program literacy and emphasizes that programs should be written in a fashion that can be read by people as well as by compilers. MAPML documents should follow a similar approach, hence, the motivation for the Meta-Information Module.

The abstract Element

| | |
|-------------------------------------|---|
| Element Name | abstract |
| Related MAPML Requirement(s) | [MAPML-Requirement-Usability-1] |
| Module | Meta-Information Module |
| Description | The <code>abstract</code> element provides a brief abstract of the pattern. It could, for example, be used by automated indexing programs. Therefore, it should be precise and minimal. Usually, the abstract would be a short-version of the description. |
| Data Type | Mixed |
| Attribute(s) | id (Optional) |
| Parent Element(s) | metadata |
| Child Element(s) | term (Optional, Cardinality = 0..Unbounded) |
| Example | <pre> <abstract> A E-Book Pattern solves the problems associated with making electronic books available on a <term>PDA </term>. It provides an <term>XML</term>-based solution to circumvent those problems. </abstract> </pre> |

Table 4.6. The abstract Element

The author Element

| | |
|-------------------------------------|--|
| Element Name | author |
| Related MAPML Requirement(s) | [MAPML-Requirement-Identifiability-2] |
| Module | Meta-Information Module |
| Description | The <code>author</code> element contains the name of the author(s) in different contexts in the pattern: pattern author, reference author, or author of the MAPML document itself. The names could be expressed in first name-last name or last name-first name forms. |
| Data Type | Singleton |
| Attribute(s) | id (Optional) |
| Parent Element(s) | metadata, reference |
| Child Element(s) | None |
| Example | <pre><!-- Different Formats for Author Names. --> <author>Monty Newborn</author> <author>Pai, Hsueh-Ieng</author></pre> |

Table 4.7. The author Element

The caption Element

| | |
|-------------------------------------|---|
| Element Name | caption |
| Related MAPML Requirement(s) | [MAPML-Requirement-Documentation] |
| Module | Meta-Information Module |
| Description | The <code>caption</code> element provides a caption for an object (such as the figure, markup, or code) included in a MAPML document. |
| Data Type | Mixed |
| Attribute(s) | None |
| Parent Element(s) | object |
| Child Element(s) | term (Optional, Cardinality = 0..Unbounded) |
| Example | <pre><object id="eb123" media-type="image/png" object-type= "figure" alternate="Snaphot of an E-Book"> <link uri="ebook.png"/> <caption> An <term>OEBPS</term> <term>E-Book</term> in</pre> |

| | |
|--|--|
| | <pre> <term>MobiPocket Reader Emulator</term>. </caption> </object> </pre> |
|--|--|

Table 4.8. The caption Element

The date Element

| | |
|-------------------------------------|--|
| Element Name | date |
| Related MAPML Requirement(s) | [MAPML-Requirement-Evolvability-2] |
| Module | Meta-Information Module |
| Description | The date element provides a timestamp (namely, pattern creation, publication, and revision) to MAPML elements. It could be used to tabulate the date(s) of evolution of the pattern. It can also be used for date of a reference. The date could be expressed in a Gregorian form of recurring day, month, and year. |
| Data Type | ISO 8601 [35] Format |
| Attribute(s) | event (Optional) |
| Parent Element(s) | metadata, reference |
| Child Element(s) | None |
| Example | <pre> <!-- The 10th Day of January of the Year 2002. --> <date event="publication">2002-01-10</date> </pre> |

Table 4.9. The date Element

The description Element

| | |
|-------------------------------------|---|
| Element Name | description |
| Related MAPML Requirement(s) | [MAPML-Requirement-Usability-1] |
| Module | Meta-Information Module |
| Description | The description element provides a synopsis of the pattern. This description is mainly for the benefit of the pattern user and should be sufficiently detailed. The description could include motivation for the pattern. |

| | |
|--------------------------|--|
| Data Type | Mixed |
| Attribute(s) | None |
| Parent Element(s) | body |
| Child Element(s) | term (Optional, Cardinality = 0..Unbounded) |
| Example | <description> The <term>E-Book</term> Pattern is motivated by the rapid ascent of electronic books that are now becoming widely-available on small devices. </description> |

Table 4.10. The description Element

The keyword Element

| | |
|-------------------------------------|---|
| Element Name | keyword |
| Related MAPML Requirement(s) | [MAPML-Requirement-Identifiability-2], [MAPML-Requirement-Navigability] |
| Module | Meta-Information Module |
| Description | Includes a sequence of term(s) relevant to the pattern. |
| Data Type | Mixed |
| Attribute(s) | None |
| Parent Element(s) | metadata |
| Child Element(s) | term (Required, Cardinality = 1..Unbounded) |
| Example | <keyword> <term>XML</term>, <term>Mobile</term> </keyword> |

Table 4.11. The keyword Element

The license Element

| | |
|-------------------------------------|---|
| Element Name | license |
| Related MAPML Requirement(s) | [MAPML-Requirement-Legality] |
| Module | Meta-Information Module |
| Description | The pattern may be associated with a license or intellectual property right statements. A license element consists of components such as copyright, license owner, and conditions for its use. It may include |

| | |
|--------------------------|---|
| | information on the type of license such as GNU Free Documentation License (GNU FDL), Open Content License, Open Publication License, and so on. |
| Data Type | Mixed |
| Attribute(s) | None |
| Parent Element(s) | metadata |
| Child Element(s) | author (Required, Cardinality = 1..Unbounded), date (Required, Cardinality = 1..Unbounded) |
| Example | <pre><license> Copyright (c) <date>2001<date>, <date>2002<date>. <author>Hsueh-Ieng Pai</author>. All Rights Reserved. Permission is granted to copy, distribute, and/or modify this document under the terms of the GNU Free Documentation License, Version 1.1 or any later version published by the Free Software Foundation. </license></pre> |

Table 4.12. The license Element

The metadata Element

| | |
|-------------------------------------|--|
| Element Name | metadata |
| Related MAPML Requirement(s) | [MAPML-Requirement-Identifiability-2] |
| Module | Meta-Information Module |
| Description | <p>The Web was originally built for human consumption and, although everything on it is machine-readable, this data is not always "machine-understandable." It is very difficult to automate tasks such as indexing, filtering, and searching on the Web.</p> <p>The solution proposed here is to use metadata ("data about data") information to describe MAPML documents published on the Web. Any information that is "about" the pattern is known as pattern metadata.</p> <p>The metadata element is a container for pattern metadata, including the pattern name, the classification scheme to which the pattern belongs to, pattern author-related information, pattern history and</p> |

| | |
|--------------------------|--|
| | current status, a brief synopsis, and keywords related to the pattern. |
| Data Type | Aggregate |
| Attribute(s) | None |
| Parent Element(s) | head |
| Child Element(s) | name (Required, Cardinality = 1), class (Required, Cardinality = 1), author (Required, Cardinality = 1..Unbounded), date (Optional, Cardinality = 0..Unbounded), abstract (Required, Cardinality = 1), license (Optional, Cardinality = 0..1), keyword (Optional, Cardinality = 0..1) |
| Example | <pre> <pattern> <head> <metadata> <!-- Include Elements for Pattern Name, Class --> <name>...</name> <date event="creation">2001-12-15</date> <abstract>...</abstract> </metadata> </head> <body> <!-- Include Other Elements. --> </body> </pattern> </pre> |

Table 4.13. The metadata Element

REMARKS

The metadata scheme in MAPML has been kept intentionally to a minimum in accordance with the traditional pattern notation. More sophisticated schemes than the above (such as Dublin Core with XML [43]) are possible.

The name Element

| | |
|-------------------------------------|---|
| Element Name | name |
| Related MAPML Requirement(s) | [MAPML-Requirement-Genericity-1], [MAPML-Requirement-Identifiability-1] |
| Module | Meta-Information Module |
| Description | The name element encapsulates the name of the pattern. A pattern should have a meaningful name that represents the problem it is addressing. A good name is vital, because it will become part of the design vocabulary. A name is usually a string of alphabetic characters. |

| | |
|--------------------------|--|
| | <p>It could be a single word or a short phrase to refer to the knowledge a pattern encompasses.</p> <p>Patterns often make connections to other available patterns, and may need to be referenced from contexts outside their scope of existence (the catalog). The pattern names alone can not guarantee globally unique identification. The hierarchical nature of our pattern classification can be useful in devising a universal naming scheme for cross-referencing. We could adopt the Java package specifying convention to serve as a resource identifier. Then, for example, for a Speech Form Pattern, we could write:</p> <p>MAPCLASS.Product.Intermediate.Non-Autonomous.Form.Speech</p> <p>or for short</p> <p>MAPCLASS.Form.Speech</p> <p>without loss of generality.</p> |
| Data Type | Singleton |
| Attribute(s) | id (Optional) |
| Parent Element(s) | metadata |
| Child Element(s) | None |
| Example | <name>MAPCLASS.E-Book</name> |

Table 4.14. The name Element

The term Element

| | |
|-------------------------------------|---|
| Element Name | term |
| Related MAPML Requirement(s) | [MAPML-Requirement-Identifiability-2] |
| Module | Meta-Information Module |
| Description | <p>The subjects of patterns, mobility, XML, Software Engineering, and the application domain in which the pattern itself exists, all have their own "vocabulary". The terms can, for example, be used in the metadata keywords, extracted to form a glossary, and so on. The <code>term</code> element encapsulates the terminology related to the pattern.</p> |

| | |
|--------------------------|--|
| Data Type | Singleton |
| Attribute(s) | term-type (Optional) |
| Parent Element(s) | abstract, caption, description, keyword, implementation, structure, strategy |
| Child Element(s) | None |
| Example | <term>IBM WebSphere Application Server</term> |

Table 4.15. The term Element

The title Element

| | |
|-------------------------------------|--|
| Element Name | title |
| Related MAPML Requirement(s) | [MAPML-Requirement-Identifiability-2] |
| Module | Meta-Information Module |
| Description | The title element provides the title of the document. This title could be used either as the title of the MAPML document or the title of a reference in the pattern. |
| Data Type | Singleton |
| Attribute(s) | None |
| Parent Element(s) | mapml, reference |
| Child Element(s) | None |
| Example | <title>Surveys on Mobility</title> |

Table 4.16. The title Element

4.5. Problem Module

The constraint Element

| | |
|-------------------------------------|----------------------------------|
| Element Name | constraint |
| Related MAPML Requirement(s) | [MAPML-Requirement-Genericity-1] |
| Module | Problem Module |

| | |
|--------------------------|---|
| Description | The constraint element describes relevant constraints (known as forces in the traditional notation) of the problem and how they interact/conflict with one another. They are used by the engineers to justify the technical decisions of their design. Constraints provide a clear picture of the complexities of the problem(s) and help defining the kinds of trade-offs that must be considered. |
| Data Type | Mixed |
| Attribute(s) | id (Optional) |
| Parent Element(s) | body |
| Child Element(s) | term (Optional, Cardinality = 0..Unbounded) |
| Example | <constraint>Limited <term>Memory</term>.</constraint> <constraint>Small Screen Size.</constraint> |

Table 4.17. The constraint Element

The context Element

| | |
|-------------------------------------|---|
| Element Name | context |
| Related MAPML Requirement(s) | [MAPML-Requirement-Genericity-1] |
| Module | Problem Module |
| Description | The context element outlines the conditions under which the problem recurs, and for which the solution is desirable. These conditions may be characteristics of the user, tasks, as well as the technical, physical, and organizational environment. Apart from the problem description, the context also provides criteria for determining when the pattern is applicable. |
| Data Type | Mixed |
| Attribute(s) | id (Optional) |
| Parent Element(s) | body |
| Child Element(s) | term (Optional, Cardinality = 0..Unbounded) |
| Example | <context> Mobile Computing Environment. Visually-Disabled User. |

| | |
|--|------------|
| | </context> |
|--|------------|

Table 4.18. The context Element

The problem Element

| | |
|-------------------------------------|--|
| Element Name | problem |
| Related MAPML Requirement(s) | [MAPML-Requirement-Genericity-1] |
| Module | Problem Module |
| Description | While constructing a mobile application, engineers can face all sorts of problems that are typically task related. The problem elements describes the problem the pattern attempts to solve within the given context and constraints of the problem. |
| Data Type | Mixed |
| Occurrence | Required |
| Attribute(s) | id (Optional) |
| Parent Element(s) | body |
| Child Element(s) | reference (Optional, Cardinality = 0..Unbounded), term (Optional, Cardinality = 0..Unbounded) |
| Example | <pre><problem> To create an <term>E-Book</term> that is <term>device independent</term>. </problem></pre> |

Table 4.19. The problem Element

4.6. Solution Module

The consequence Element

| | |
|-------------------------------------|--|
| Element Name | consequence |
| Related MAPML Requirement(s) | [MAPML-Requirement-Genericity-1] |
| Module | Solution Module |
| Description | The consequence element describes impact and trade-offs from the |

| | |
|--------------------------|--|
| | application of the pattern. It is likely that a pattern may improve one aspect at the cost of deteriorating others. In general, this section focuses on the results of using a particular pattern, and notes the pros (such as, what aspects have been improved) and cons (such as, what aspects have worsened) that may result from the application of the pattern. |
| Data Type | Mixed |
| Attribute(s) | impact (Required) |
| Parent Element(s) | body |
| Child Element(s) | term (Optional, Cardinality = 0..Unbounded) |
| Example | <pre> <consequence impact="positive"> Provides instant access to different parts of the book. </consequence> <consequence impact="negative"> Reader oriented towards paper-based books or unfamiliar with the linking mechanism may experience a learning curve. </consequence> </pre> |

Table 4.20. The consequence Element

The implementation Element

| | |
|-------------------------------------|---|
| Element Name | implementation |
| Related MAPML Requirement(s) | [MAPML-Requirement-Usability-2] |
| Module | Solution Module |
| Description | The implementation element includes an actual implementation of the solution suggested by the pattern. Mobile application patterns, upon implementation, can exist in forms of various objects -- figure, markup, or code -- that explain how the pattern can be implemented. |
| Data Type | Mixed |
| Occurrence | Required |
| Attribute(s) | id (Optional) |
| Parent Element(s) | strategy |
| Child Element(s) | object (Optional, Cardinality = 0..Unbounded), term (Optional, Cardinality = 0..Unbounded) |

| | |
|----------------|---|
| Example | <pre> <implementation> <object id="eb456" object-type="markup" media-type= "text/xml"> <oeb:package unique-identifier="uid"> <oeb:metadata> <!-- Dublin Core Metadata. --> </oeb:metadata> <oeb:manifest> <!-- List of E-Book Files. --> </oeb:manifest> <oeb:spine> <!-- Linear Reading Order. --> <oeb:itemref idref="Cover"/> </oeb:spine> </oeb:package> </object> </implementation> </pre> |
|----------------|---|

Table 4.21. The implementation Element

The object Element

| | |
|-------------------------------------|--|
| Element Name | object |
| Related MAPML Requirement(s) | [MAPML-Requirement-Extensibility] |
| Module | Solution Module |
| Description | <p>A MAPML document can include a variety of external objects to assist with the pattern solution or to describe a scenario. The <code>object</code> element is the container for the possible objects: figure, markup, or code.</p> <p>Contents of both the <code>markup</code> and <code>code</code> are to be presented as listings -- it is not required that either the markup or code be processed. When a MAPML document is presented to a user agent, it may process the markup (if it supports the corresponding vocabulary) and include the results.</p> |
| Data Type | Mixed |
| Occurrence | Required |
| Attribute(s) | id (Optional), media-type (Required), object-type (Required), alternate (Conditional; Required if the object-type is a figure) |
| Parent Element(s) | implementation, scenario, structure |
| Child Element(s) | caption (Optional, Cardinality = 0..1), link (Conditional; Required if the object referenced is external to the MAPML document, Cardinality |

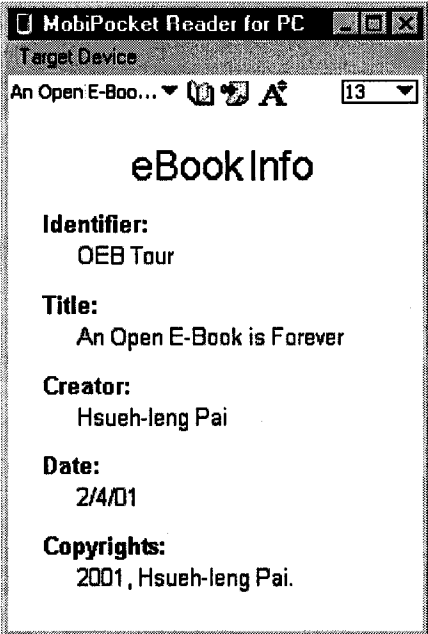
| | |
|----------------|---|
| | = 0..1) |
| Example | <pre> <object object-type="figure" media-type="image/png" alternate="Snapshot of an E-Book"> <link uri="ebook.png"/> <caption> An <term term-type="abbreviation">OEBPS</term> <term>E-Book</term> in <term term-type="tool">MobiPocket Reader Emulator</term>. </caption> </object> </pre> <p>This object may appear on rendering as</p>  <p>The screenshot shows a window titled 'MobiPocket Reader for PC'. Inside, there's a 'Target Device' section with a dropdown menu showing 'An Open E-Book...' and a page number '13'. Below this is a section titled 'eBookInfo' containing the following details:</p> <ul style="list-style-type: none"> Identifier: OEB Tour Title: An Open E-Book is Forever Creator: Hsueh-leng Pai Date: 2/4/01 Copyrights: 2001, Hsueh-leng Pai. |

Table 4.22. The object Element

The rationale Element

| | |
|-------------------------------------|---|
| Element Name | rationale |
| Related MAPML Requirement(s) | [MAPML-Requirement-Usability-1] |
| Module | Solution Module |
| Description | Given a problem and a large collection of patterns, one faces the issue of making a viable choice. The <code>rationale</code> element describes, in a solution-independent manner, the reasoning behind and suitability of the pattern as a justified choice towards solving the problem. The |

| | |
|--------------------------|--|
| | rationale assists an engineer in making the appropriate choice by describing how and why the pattern works, with an insight into the internal structure and key mechanisms of the system. |
| Data Type | Mixed |
| Attribute(s) | None |
| Parent Element(s) | body |
| Child Element(s) | term (Optional, Cardinality = 0..Unbounded) |
| Example | <pre> <pattern> <!-- Other Elements --> <name>MAPCLASS.E-Book</name> <solution> <rationale> The rationale for presenting a solution based on <term>XML</term> is that it has several advantages. By the application of the single-source approach, it is possible to author documents once and serve them everywhere. </rationale> </solution> </pattern> </pre> |

Table 4.23. The rationale Element

The scenario Element

| | |
|-------------------------------------|--|
| Element Name | Scenario |
| Related MAPML Requirement(s) | [MAPML-Requirement-Genericity-1] |
| Module | Solution Module |
| Description | <p>The scenario element gives instance(s) of "real-world" situations where the specified pattern has been used by the author and/or by other users. The section may also include counter examples -- examples of cases where the pattern should have been used but was not, and "non-examples" -- examples of cases where the pattern should not have been used but was.</p> <p>The examples included in this section help engineers to understand the scope and domain of applicability of the pattern. They also enforce</p> |

| | |
|--------------------------|--|
| | <p>the fact that the pattern describes a proven solution. This is crucial in judging the viability of and quantifying the actual use of the pattern.</p> <p>The example(s) can be provided in several ways: prose, figure, markup, and so on that illustrate the use of the pattern.</p> |
| Data Type | Mixed |
| Occurrence | Required |
| Attribute(s) | None |
| Parent Element(s) | Body |
| Child Element(s) | object (Optional, Cardinality = 0..Unbounded), term (Optional, Cardinality = 0..Unbounded) |
| Example | <pre><scenario> The <term>E-Book</term>s are used by established book distribution enterprises (such as, Amazon.com, and Barnes and Noble) and by noted publishers (such as, Addison-Wesley). </scenario></pre> |

Table 4.24. The scenario Element

The solution Element

| | |
|-------------------------------------|---|
| Element Name | solution |
| Related MAPML Requirement(s) | [MAPML-Requirement-Genericity-1] |
| Module | Solution Module |
| Description | <p>The solution element includes a description of the actual solution provided by the pattern to solve the problem. It describes the solution approach briefly and the solution aspects themselves in detail. The solution aspects identify the pattern's structure, presentation, logic, and behavior.</p> |
| Data Type | Mixed |
| Attribute(s) | None |
| Parent Element(s) | body |
| Child Element(s) | reference (Optional, Cardinality = 0..Unbounded), structure (Required, Cardinality = 1), strategy (Required, Cardinality = 1) |
| Example | <pre><solution> <!-- Structure and Strategy --></pre> |

| | |
|--|-------------|
| | </solution> |
|--|-------------|

Table 4.25. The solution Element

The strategy Element

| | |
|-------------------------------------|---|
| Element Name | strategy |
| Related MAPML Requirement(s) | [MAPML-Requirement-Usability-2] |
| Module | Solution Module |
| Description | Engineers discover and invent new ways to implement the pattern, producing new strategies for well-known patterns. To accommodate that, strategies provide an extensibility point for each pattern. The strategy element includes a description of different ways a pattern can be implemented. |
| Data Type | Mixed |
| Attribute(s) | None |
| Parent Element(s) | solution |
| Child Element(s) | implementation (Required, Cardinality = 1..Unbounded), term (Optional, Cardinality = 0..Unbounded) |
| Example | <pre><strategy> The tree structure of an <term>E-Book</term> can be implemented in several ways. The most reader-friendly way is to follow the format of a regular book. There is a cover page, table of contents, and a set of chapters. The items in the table of contents link to preface, individual chapters, and sections. ... <implementation>...</implementation> </strategy></pre> |

Table 4.26. The strategy Element

The structure Element

| | |
|-------------------------------------|---|
| Element Name | structure |
| Related MAPML Requirement(s) | [MAPML-Requirement-Usability-2] |
| Module | Solution Module |
| Description | The structure element includes a description of a pattern at a high |

| | |
|--------------------------|--|
| | level abstraction. This could be done by using prose or by using a visual modeling notation. For example, use of the UML Diagrams can be made to show the basic structure and data flow of the solution. |
| Data Type | Mixed |
| Attribute(s) | None |
| Parent Element(s) | solution |
| Child Element(s) | object (Optional, Cardinality = 0..Unbounded), term (Optional, Cardinality = 0..Unbounded) |
| Example | <pre><structure> The solution structure of MAPCLASS.E-Book is a tree where all the nodes are accessible from the root. </structure></pre> |

Table 4.27. The structure Element

4.7. Structure Module

The body Element

| | |
|-------------------------------------|--|
| Element Name | body |
| Related MAPML Requirement(s) | [MAPML-Requirement-Navigability], [MAPML-Requirement-Processability-1], [MAPML-Requirement-Usability-1] |
| Module | Structure Module |
| Description | The body element is the container that holds the actual content of a mobile application pattern. |
| Data Type | Aggregate |
| Occurrence | Required |
| Attribute(s) | None |
| Parent Element(s) | pattern |
| Child Element(s) | description (Required, Cardinality = 1), context (Required, Cardinality = 1), problem (Required, Cardinality = 1), constraint (Required, Cardinality = 1..Unbounded), rationale (Required, Cardinality = 1), solution (Required, Cardinality = 1), consequence (Required, Cardinality = 1..Unbounded), pattern.related (Required, Cardinality = 1..Unbounded), scenario (Required, Cardinality = 1..Unbounded) |

| | |
|----------------|---|
| Example | <pre><body> <!-- Other Elements Here. --> </body></pre> |
|----------------|---|

Table 4.28. The body Element

The head Element

| | |
|-------------------------------------|---|
| Element Name | head |
| Related MAPML Requirement(s) | [MAPML-Requirement-Navigability], [MAPML-Requirement-Processability-1], [MAPML-Requirement-Usability-1] |
| Module | Structure Module |
| Description | The head element is a container of information that is not directly related to the pattern. It provides a separation of pattern core from pattern metadata. |
| Data Type | Aggregate |
| Attribute(s) | None |
| Parent Element(s) | pattern |
| Child Element(s) | metadata (Required, Cardinality = 1) |
| Example | <pre><head> <metadata> <!-- Other Elements Here. --> </metadata> </head></pre> |

Table 4.29. The head Element

The mapml Element

| | |
|-------------------------------------|---|
| Element Name | mapml |
| Related MAPML Requirement(s) | [MAPML-Requirement-Genericity-1] |
| Module | Structure Module |
| Description | The mapml element is the root element of a MAPML document. It can contain one or more pattern elements. |
| Data Type | Aggregate |
| Attribute(s) | id (Optional), version (Required), xml:lang (Required), xml:space |

| | |
|--------------------------|--|
| | (Optional) |
| Parent Element(s) | None |
| Child Element(s) | pattern (Required, Cardinality = 1..Unbounded), title (Required, Cardinality = 1) |
| Example | <pre><mapml version="1.0" xml:lang="en"> <pattern> <!-- Other Elements Here. --> </pattern> </mapml></pre> |

Table 4.30. The mapml Element

The pattern Element

| | |
|------------------------------|---|
| Element Name | pattern |
| Related MAPML Requirement(s) | [MAPML-Requirement-Genericity-1], [MAPML-Requirement-Identifiability-1], [MAPML-Requirement-Processability-1], [MAPML-Requirement-Usability-1] |
| Module | Structure Module |
| Description | The pattern element is a container for pattern information organized into head and body. |
| Data Type | Mixed |
| Attribute(s) | id (Required), version (Required) |
| Parent Element(s) | mapml |
| Child Element(s) | head (Required, Cardinality = 1), body (Required, Cardinality = 1) |
| Example | <pre><pattern> <head> <!-- Other Elements Here. --> </head> </body> <!-- Other Elements Here. --> <body> </body> </pattern></pre> |

Table 4.31. The pattern Element

4.8. MAPML Attribute Definitions

MAPML defines ten attributes: alternate, event, id, impact, media-type, object-type, relation, term-type, uri, and version. They are described in the following tables.

MAPML “borrows” two attributes from the XML 1.0 Specification [55]: `xml:lang` and `xml:space`. `xml:lang` is used to indicate the natural language being used in the document, `xml:space` gives directions to the processor for controlling white space. MAPML also uses an attribute from Namespaces in XML Specification [49], `xmlns`, to uniquely identify its elements and attributes.

A “|” indicates an exclusive or.

The alternate Attribute

| | |
|-------------------------------------|--|
| Attribute Name | alternate |
| Related MAPML Requirement(s) | [MAPML-Requirement-Usability-2] |
| Description | The purpose of the <code>alternate</code> attribute is to make objects other than plain text accessible. It provides a short text description for non-textual objects included in MAPML documents. |
| Data Type | Character Data |
| Related Element(s) | object |

Table 4.32. The alternate Attribute

The event Attribute

| | |
|-------------------------------------|---|
| Attribute Name | event |
| Related MAPML Requirement(s) | [MAPML-Requirement-Evolvability-2] |
| Description | The <code>event</code> attribute provides the context of timestamping. For example, the context could be related to the evolution -- creation, publication or modification -- of the pattern. |
| Data Type | creation publication revision |
| Related Element(s) | date |

Table 4.33. The event Attribute

The `id` Attribute

| | |
|-------------------------------------|---|
| Attribute Name | <code>id</code> |
| Related MAPML Requirement(s) | [MAPML-Requirement-Identifiability-1] |
| Description | The <code>id</code> attribute uniquely identifies an element within a document. Its value is an XML identifier. |
| Data Type | XML ID. For acceptable values, see the XML 1.0 Specification [55]. |
| Related Element(s) | implementation, mapml, pattern, reference |

Table 4.34. The `id` Attribute

The `impact` Attribute

| | |
|-------------------------------------|---|
| Attribute Name | <code>impact</code> |
| Related MAPML Requirement(s) | [MAPML-Requirement-Genericity-1] |
| Description | The <code>impact</code> attribute provides the type of consequence that a pattern can have when applied to a given situation. |
| Data Type | positive negative |
| Related Element(s) | consequence |

Table 4.35. The `impact` Attribute

The `media-type` Attribute

| | |
|-------------------------------------|---|
| Attribute Name | <code>media-type</code> |
| Related MAPML Requirement(s) | [MAPML-Requirement-Processability-1] |
| Description | The purpose of the <code>media-type</code> attribute is to provide user agents clue of the data format of the external object being included in a MAPML document. |
| Data Type | MIME (Internet Media Type). For acceptable values, see the IETF RFC 2045 [15]. |

| | |
|---------------------------|--------|
| Related Element(s) | object |
|---------------------------|--------|

Table 4.36. The media-type Attribute

The object-type Attribute

| | |
|-------------------------------------|--|
| Attribute Name | object-type |
| Related MAPML Requirement(s) | [MAPML-Requirement-Extensibility] |
| Description | The object-type attribute states the types of object included in the MAPML document. The possibilities are: figure, markup, or code. A figure can be used to describe the structure of the pattern solution, the pattern implementation, or the pattern scenario. Markup provides an implementation of the pattern solution in form of a complete document or fragment based on a markup language, preferably based on XML. A snippet of code (script, or a style-sheet, or a program) provides an implementation of the pattern solution based on a formal (non-markup) language. |
| Data Type | figure markup code |
| Related Element(s) | object |

Table 4.37. The object-type Attribute

The relation Attribute

| | |
|-------------------------------------|---|
| Attribute Name | relation |
| Related MAPML Requirement(s) | [MAPML-Requirement-Genericity-1] |
| Description | <p>The relation attribute symbolizes the type of relationship that the related pattern has with the pattern under study.</p> <p>The relationship can be categorized as superordinate, subordinate, sibling, or competitor. A superordinate pattern is the super set of the described pattern. It can therefore contain the described pattern and</p> |

| | |
|---------------------------|---|
| | possibly other patterns. A subordinate pattern is a subset of (that is, it can be embedded into) the described pattern. It is therefore a part of the described pattern. A sibling pattern belongs to the same pattern category as the described pattern. It provides either replaceable or enhanced function to the described pattern, but not necessarily in the same context. Finally, a competitor pattern can provide the identical or similar function as the described pattern. Thus, it can replace the described pattern in the same context. |
| Data Type | superordinate subordinate sibling competitor |
| Related Element(s) | pattern.related |

Table 4.38. The relation Attribute

The term-type Attribute

| | |
|-------------------------------------|--|
| Attribute Name | term-type |
| Related MAPML Requirement(s) | [MAPML-Requirement-Processability-2] |
| Description | The term-type attribute symbolizes the nature of the term that can exist if various forms. |
| Data Type | abbreviation concept principle technology tool |
| Related Element(s) | term |

Table 4.39. The term-type Attribute

The uri Attribute

| | |
|-------------------------------------|--|
| Attribute Name | uri |
| Related MAPML Requirement(s) | [MAPML-Requirement-Navigability] |
| Description | The uri attribute provides the URI of the content it is associated with. |
| Data Type | URI. For acceptable values, see IETF RFC 2396 [7]. |
| Related Element(s) | link |

Table 4.40. The uri Attribute

The version Attribute

| | |
|-------------------------------------|---|
| Attribute Name | version |
| Related MAPML Requirement(s) | [MAPML-Requirement-Evolvability-2] |
| Description | The version attribute provides a numerical value of the release date of either the MAPML document or the pattern. |
| Data Type | decimal |
| Related Element(s) | mapml, pattern |

Table 4.41. The version Attribute

4.9. MAPML Identification

MAPML documents need to be identified by user agents and processors. This section describes the facilities provided by MAPML in that direction.

4.9.1. MAPML Internet Media (MIME) Type

In accordance with IETF RFC 3023 [34], Appendix A.15, MAPML documents should be served as the media type

```
application/mapml+xml
```

Until user agents recognize the '+xml' suffix for XML-based MIME types, MAPML documents could be served as the media type `text/xml` (IETF RFC 3023, Appendix A.1).

4.9.2. MAPML Namespace

The XML Namespace assigned to MAPML is

<http://www.cs.mcgill.ca/mapml>

The prefix `mapml:` is used by convention to denote the MAPML Namespace, although any prefix can be used. This is necessary when authoring and delivering MAPML documents, particularly those that contain non-MAPML markup fragments.

4.9.3. MAPML Filename Extension

The use of `mapml` as the filename extension for MAPML documents is recommended.

4.10. MAPML Conformance

This section provides the desirable criteria for conformance of a MAPML document and that of a MAPML processor.

4.10.1. MAPML Document Conformance

A MAPML conforming document must satisfy the following criteria:

1. It must conform to the XML 1.0 Specification. In particular, it must be well-formed, as defined in the Section 2.1 of the XML 1.0 Specification [55].
2. It must specify a character encoding in the XML processing instruction.
3. It must declare `mapml` as its root element.
4. It must contain at least one `pattern` element.
5. It must validate against the MAPML Schema (Appendix A).
6. If any namespaces other than MAPML are used in the document, it must conform to the Namespaces in XML [49]. In particular, if it includes any fragments of non-MAPML markup, it must specify the MAPML Namespace URI in its root element and prefix all MAPML elements with "`mapml:`".
7. If it makes any use of CSS, it shall conform to Cascading Style Sheets, Level 2 Specification [47].

8. If it makes any references to external style sheets, it shall conform to Associating Style-sheets with XML Documents [52].

4.10.2. MAPML Processor Conformance

A MAPML processor must be an conforming XML processor as defined in Section 5 of the XML 1.0 Specification [55].

4.11. Summary

MAPML provides several features to capture mobile application patterns. These features are expressed as elements and attributes, and organized in form of modules.

To perform certain desirable tasks such as authoring, presenting, processing, and validating, various tools are required. Chapter 5 gives an overview of MAPML Utilities (MAPML-UTIL).

Chapter 5

MAPML Utilities (MAPML-UTIL)

This chapter presents a collection of utilities for validating, authoring, processing, and presenting MAPML documents. These utilities can help a pattern author or user in carrying out various tasks required to create quality patterns. MAPML-UTIL is based on technologies and tools that are available freely and/or as Open Source. Figure 5.1 provides a schematic.

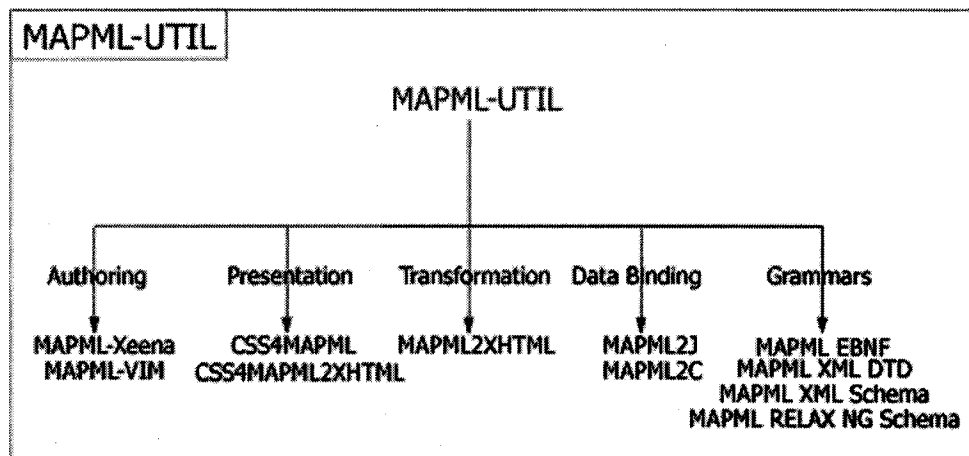


Figure 5.1. MAPML Family of Utilities (MAPML-UTIL).

5.1. MAPML Grammars

A formal grammar provides rules for syntax, structure, and data types of MAPML documents. It provides a means to validate MAPML documents for conformance.

We have included grammar descriptions in EBNF, XML DTD, XML Schema, and RELAX NG. Among these, XML Schema and RELAX NG belong to the family of XML Schema Languages -- grammar languages of XML vocabularies in XML syntax.

The rationale to provide more than one grammar description is that each has its own advantages and disadvantages [38]

5.1.1. EBNF Implementation for MAPML

Extended Backus-Naur Form (EBNF) [27] is a notation for syntax specification of formal (context-free) languages. It is a successor of BNF that is semantically equivalent, more readable, and more succinct. EBNF is ISO/IEC 14977 Standard. It provides the advantage of automatically generating parsers and syntax checkers. However, EBNF provides only a limited support for data types and does not provide explicit support for some other desirables in an XML document such as character entities.

We have created an EBNF grammar for MAPML, and used that to generate a basic MAPML parser in C language.

5.1.2. XML DTD Implementation for MAPML

SGML Document Type Definition (DTD) is an ISO/IEC Standard. XML DTD is based on SGML DTD and is defined in the XML 1.0 Specification. The advantages of XML DTD are that it is simple, leverages on the experience of a broad user-base, and has a rich tool support. XML DTDs provide support for character entities. However, XML DTDs have some limitations. For example, they do not support XML namespaces explicitly, do not support type inheritance, and have very limited support for data types.

An XML DTD implementation for MAPML has been created. The MAPML DTD was validated using various means (including the DTDParser, Tibco Extensibility TurboXML, and XML Spy) to ensure its correctness and robustness.

REMARKS

An early attempt to provide an XML DTD for patterns was made in [65]. However, the details are sketchy and the DTD syntax seems to be incorrect. A collection of patterns in user interface design [68] also provides an XML DTD for its pattern collection. However, the low level of granularity of the grammar makes it less expressive, and hence it will not be able to capture all aspects of patterns in a mobile setting. These

efforts evidently also possess the inherent limitations of the DTD. Furthermore, the documents based on the DTDs mix structure and presentation.

5.1.3. XML Schema Implementation for MAPML

XML Schema [59] is a schema language for XML developed under the auspices of W3C. It offers facilities for describing the structure and constraining the contents of XML documents. The advantages of XML Schema are that it supports XML namespaces, provides a rich library of data types, facilitates inclusion of user-defined types, and is expressed in XML syntax making it useful to the existing XML software-base. However, XML Schema has some limitations. For example, it does not support entities, and does not allow the relationships between the values of different attributes and contents of elements to be validated.

An XML Schema implementation for MAPML has been created. (See Appendix A.) The MAPML XML Schema was validated with the W3C XML Schema Validator and Tibco Extensibility TurboXML, and checked for quality with the IBM XML Schema Quality Checker.

5.1.4. RELAX NG Implementation for MAPML

REgular LAnguage for XML Next Generation (RELAX NG) [41] is a schema language for XML developed under the auspices of OASIS. It focuses upon description and validation of the structure and content of an XML document without attempting to specify application processing semantics (interpretation). RELAX NG offers a middle ground between XML DTDs and XML Schema. The advantages of RELAX NG are that it is simple in design, has a shallow learning curve, is expressed in XML syntax, does not change the information set of an XML document, supports XML namespaces, treats attributes uniformly with elements as far as possible, has unrestricted support for unordered content, has unrestricted support for mixed content, and can partner with a separate data typing library. However, RELAX NG places certain restrictions in trade-off for simplicity. For example, it does not allow defaults for attributes to be specified, does not allow entities to be specified, and does not specify whether white space is significant.

We have created a RELAX NG implementation for MAPML. MAPML RELAX NG Schema was validated with Jing, a RELAX NG validator in Java, and checked with the RELAX NG Verifier.

5.2. Authoring MAPML Documents

Any text editor can be used to author MAPML documents. In particular, Emacs (or its variants) using an XML mode should suffice. However, syntax-sensitive editing provides several advantages. In this section, we discuss two editing solutions that we have configured specifically for MAPML documents.

5.2.1. MAPML with XEENA

Xeena [21] is a visual XML editor in Java for editing valid XML documents derived from any valid XML DTD or XML Schema. The editor takes as input a given XML DTD or XML Schema, and automatically builds a palette containing the elements defined in the XML DTD or XML Schema. A key feature of Xeena is its syntax directed editing ability. Xeena is aware of the XML DTD or XML Schema. By making only authorized elements icons active, it automatically ensures that all documents generated are valid according to the given XML DTD or XML Schema.

We have created a basic **MAPML DTD Profile** so that Xeena can be readily used with MAPML DTD and MAPML documents. Figure 5.2 illustrates the Xeena Interface with this profile.

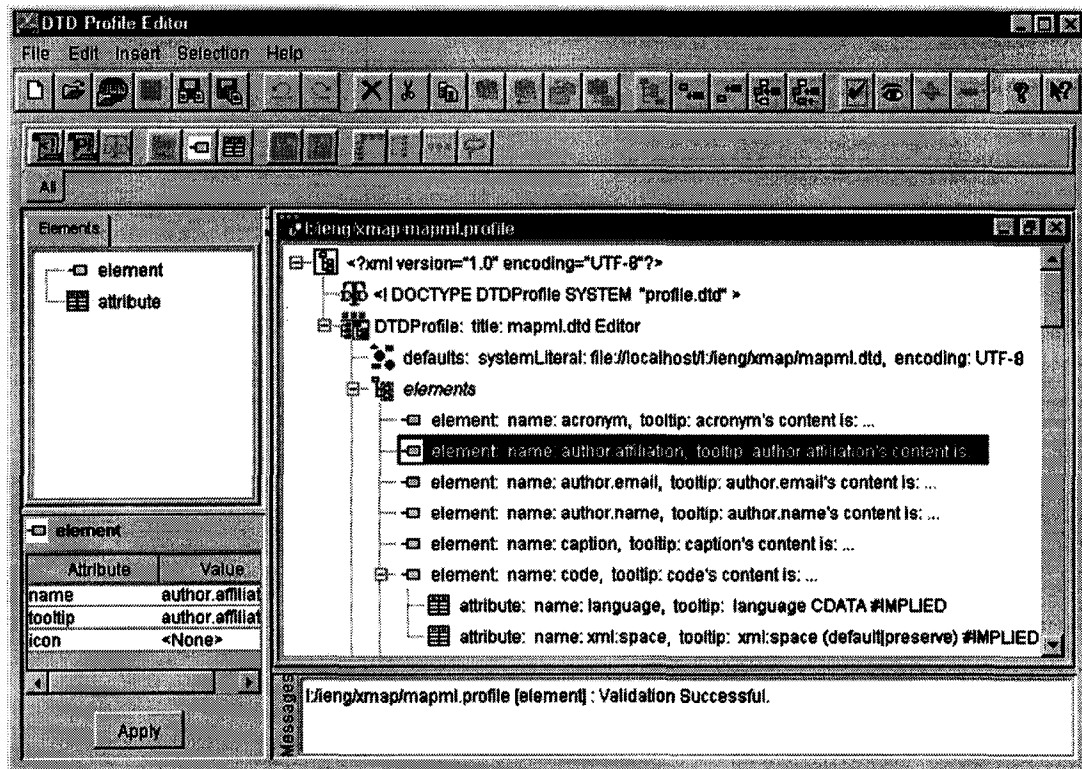


Figure 5.2. Xena-MAPML DTD Interface.

5.2.2. MAPML with VIM

VIM (VI "Improved") editor is a major improvement of the UNIX standard text editor Vi that is available for many platforms including MacOS, VMS, Windows, and various derivatives of UNIX and Linux. VIM adds multi-level undo, syntax highlighting, command line history, filename completion, block operations, GUI support, and many more features.

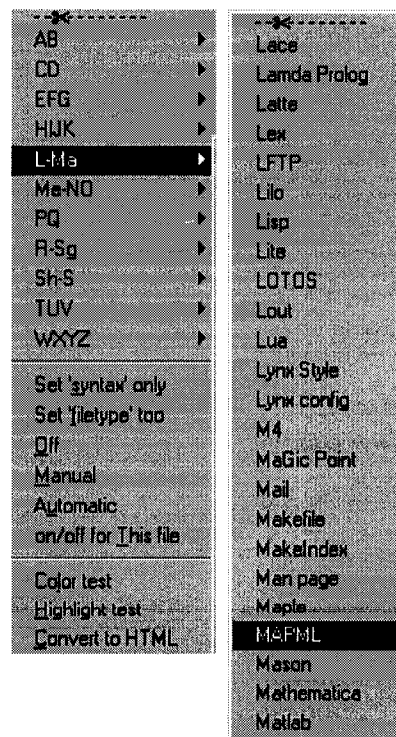
We have created a **VIM Syntax File** for MAPML that could be useful for authoring mobile patterns. Figure 5.3 shows a sample MAPML document in the VIM Interface with the MAPML Syntax File.

```

<?xml version = "1.0"?>
<mapml version = "string" xml:lang = "name" xml:space = "optional">
<!--(pattern+)-->
<pattern id = "required">
<!--(pattern.name , pattern.alias? , pattern.head , pattern.body)-->
<pattern.name>
<!--(text , link)-->
<text>only text</text>
<link uri = "required">only text</link>
</pattern.name>
<pattern.alias>only text</pattern.alias>
<pattern.head>
<!--(pattern.metadata)-->
<pattern.metadata>
<!--(text* , pattern.author+ , date , pattern.version? , pattern.abstract
, pattern.license? , pattern.keywords)-->
<text>only text</text>
<pattern.author>
<!--(author.name , author.affiliation? , author.email)-->
<author.name>only text</author.name>
<author.affiliation>only text</author.affiliation>
<author.email>only text</author.email>

```

(a) A MAPML Document in the VIM Editor.



(b) Location of the MAPML Syntax Option in the VIM Interface Syntax Menu

Figure 5.3. MAPML Syntax File in the VIM Interface.

5.3. Processing MAPML Documents

For processing XML documents, there are two major approaches: event-based processing and tree-based processing. In an event-based approach, the processing is sequential and an XML document is treated like a text stream (a string of characters). When an "event" (say, start/end of an element) is "discovered", it triggers a function call (to an event handler). This callback mechanism is similar to that of a GUI's event handling. There is no need to cache the entire document in-memory or secondary storage. The limitation of this approach is that it is resource intensive when the structure is very hierarchical (tree-oriented). In a tree-based approach, the processing is hierarchical and an XML document is treated as an in-memory tree of nodes (objects). The limitation of this approach is that in-memory data structures are resource intensive when the document size is large.

5.3.1. XML APIs and MAPML

To standardize the implementations of event-callback and tree-construction-and-traversal mechanisms, several XML Application Programming Interfaces (APIs) have been developed. Simple API for XML (SAX) [11] is widely-used example of the former case, and Document Object Model (DOM) [56] of the latter case. There are also some APIs, such as JDOM [10], that are based on a hybrid model of the two. There are several parsers in a variety of programming languages are available today that implement these APIs and can be used to process MAPML documents.

5.3.2. MAPML and Data Binding

Lower level XML APIs like SAX, DOM, and JDOM provide **generic views** of an XML document. However, a document that represents a pattern description is most easily worked with using methods "tailored" for it. For example, it is preferable to use methods such as `getPattern(String problem)`, rather than `getElement(String name)` and `setContent(String content)`. This can be accomplished by **data binding**.

XML Data Binding [8] is mapping an instance of an XML grammar into the appropriate object model (set of classes and types which represent the data). That is, with XML data binding, XML grammar definitions (such as, XML DTD or XML Schema) can be automatically translated into programming language code (say, in C++ or Java).

The data binding facility can significantly improve the performance and functionality of server-based programs and other applications that process XML, while at the same time reducing both development and maintenance costs. The generated classes include the code required to validate data content as well as data structure, thus relieving the programmer from the necessity of doing so. These classes are "lightweight" in the sense that they carry no unnecessary functionality. As a result, data binding applications will use a minimum amount of memory and run as efficiently as possible. XML data binding allows applications to manipulate content that has been serialized as XML in a way that is more natural than using the general-purpose APIs. For example, an application that uses the generated classes can build a Java object tree representing an XML document, manipulate the content of the tree, and re-generate XML documents from the tree -- all without requiring the developer to write complex parsing and processing code. The generated application runs with a speed comparable to that of a SAX application, and builds an in-memory data structure similar (but without the additional overhead intrinsic) to a DOM. The use of data binding, coupled with high-performance virtual machines like the Java HotSpot Virtual Machine, makes it possible to deliver and maintain high-performance XML-processing applications with minimum development effort. If the underlying XML grammar changes, all that is required is a "recompilation" to generate the classes again, without making modifications to the underlying application that uses those classes. This reduces maintenance costs as well as the original programming effort.

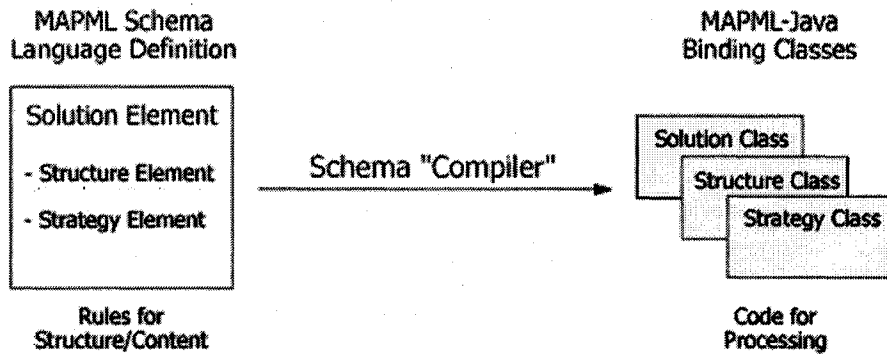


Figure 5.4. MAPML Data Binding Process.

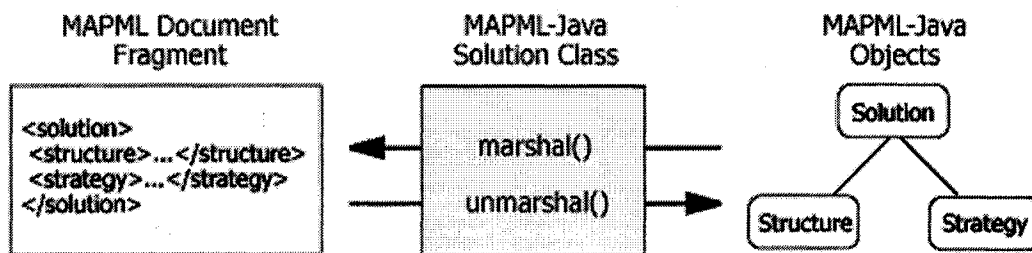


Figure 5.5. Marshalling and Un-Marshalling MAPML Documents.

We have created MAPML-Java Data Binding (MAPML2J) and MAPML-C++ Data Binding (MAPML2C). Figure 5.4 shows how data binding operates by "compiling" a schema specification to produce Java classes. Figure 5.5 shows the process of marshalling (converting an in-memory structure of objects to an XML data stream) and un-marshalling (converting an XML data stream to an in-memory structure of objects) between MAPML documents and MAPML-Java objects.

REMARKS

- The code that is automatically generated via XML data binding should be used with care [2]. This is because neither XML DTDs nor XML Schemas have a complete one-to-one mapping with various object-oriented programming language concepts. Although mapping of simple concepts, such as parent-child associations with aggregations is possible, concepts such as derivation by restriction do not have analogs in programming languages. Thus, XML data

binding should be viewed as a useful complement, but not a replacement of application-specific low-level programming code.

5.4. Presenting MAPML Documents

MAPML, by applying the principle of Separation of Qualities, separates structure from presentation. As a result, it does not provide any presentation semantics. The task of rendering MAPML documents is relegated to style sheet languages.

The style sheet approach has several advantages towards document engineering. They enable documents to remain vendor, platform, and device independent. The style sheets themselves are also vendor and platform independent. Referring to style sheets from documents can simplify maintenance and retain consistent look-and-feel throughout a pattern catalog collection. In case of any modifications, only the style sheet file(s) needs to be changed. It is also easy to change the style sheet with little or no impact on the markup.

There are two approaches for using style sheets with MAPML documents:

1. **Direct Presentation.** Presentation of MAPML documents natively.
2. **Indirect Presentation.** Transformation of MAPML documents to a presentational format.

Figure 5.6 illustrates the two approaches, which we describe in detail below.

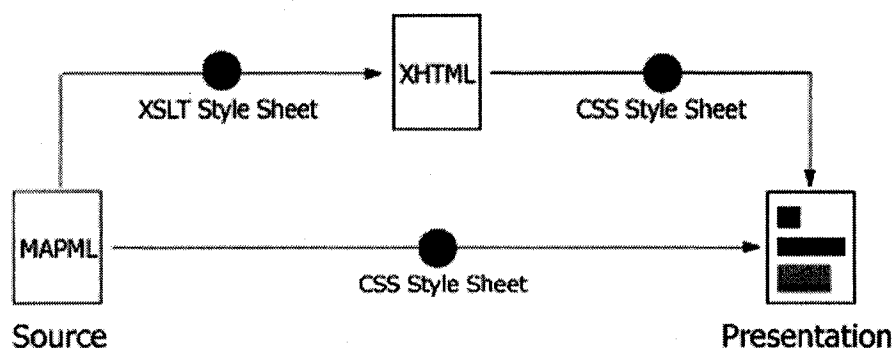


Figure 5.6. The Two Approaches for Presentation of MAPML Documents.

5.4.1. Direct Presentation of MAPML Documents

MAPML documents can be presented directly in a generic XML-aware user agent that supports a style sheet language. We have chosen Cascading Style Sheets (CSS) [47] as our style sheet language. CSS is a simple style language for structured documents, and is machine processable, human readable, and writable. It associates style with the nodes of a MAPML document tree, but does not alter the structure or content of the document itself.

The two key characteristics of CSS that are of interest to us are **cascading** and **inheritance**. CSS allows more than one style sheet from multiple sources (author, user, and user agent) to be associated with and influence the presentation of a single document. This feature is known as cascading where the different style sheets are viewed as being applied in a sequence. A tree of MAPML elements can inherit stylistic properties. Through inheritance, CSS property values set on one element will be transferred down the tree to its descendants. This enables style sheets to become simpler and more efficient (shorter).

We have created a simple CSS style sheet for MAPML documents. The basic idea is as follows. A CSS style sheet consists of a list of rules. A rule is a combination of a selector and a declaration block. A selector is the connection between the XML document and style. A declaration consists of a property and its value. The declarations are grouped within a block enclosed by curly braces ({...}). Each style property in a declaration starts with the property's name, then a colon (:), and lastly the value for this property. Figure 5.7 shows the anatomy of a CSS style sheet.

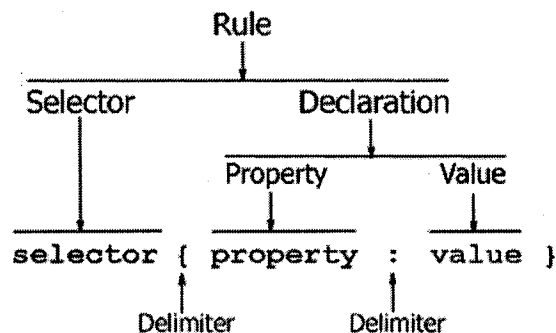


Figure 5.7. Internal Structure of a CSS Style Sheet.

Now, style rules can be associated with MAPML elements by using them as selectors and attaching desired property-value pairs. This is illustrated in the following fragment.

```
pattern.problem {  
  width      : 100%;  
  border     : solid 1px rgb(000,000,000);  
  background-color : #cef;  
  padding    : 2px;  
}
```

When associated with the CSS style sheet, MAPML documents can be directly presented in a XML/CSS-aware user agent, such as Amaya, Opera, Microsoft Internet Explorer, and Mozilla.

5.4.2. Indirect Presentation of MAPML Documents Transformed to XHTML

MAPML documents could be (down) transformed to a format oriented towards presentational purposes. This is especially useful as an alternative to user agents that do not support XML or those with constrained capabilities, to support legacy user agents, or to add extra presentational capabilities. For this purpose, we have adopted XHTML Basic as the language of choice for presentation as it is simple, has adequate features to present the descriptive elements of MAPML, and is applicable to devices in both Mobile and Web environments. (With minor changes to the grammar declaration, MAPML documents can also be easily transformed to XHTML 1.1.) As the transformation language, we have chosen XSL Transformations (XSLT) [52]. XSLT is a high-level declarative language designed for transforming the structure of XML documents. It allows the result tree to be serialized as XML as well as non-XML.

We have created an XSLT style sheet that transforms arbitrary MAPML documents to XHTML Basic documents. The most critical aspect of the transformation is the mapping of elements in MAPML to those in XHTML Basic. For example, the MAPML `text` element could be mapped to a `p` (paragraph) element in XHTML. This mapping is not unique and there is (an obvious but not unacceptable) information loss using this approach as the structure of the source document can be altered. Figure 5.8

shows a simplified version of a scenario where a MAPML document is mapped to an XHTML Basic document using XSLT. The resulting XHTML Basic document is associated with a CSS style sheet and can be viewed in a XHTML Basic-compliant user agent, such as Amaya, Nokia Mobile Browser and Openwave Mobile Browser.

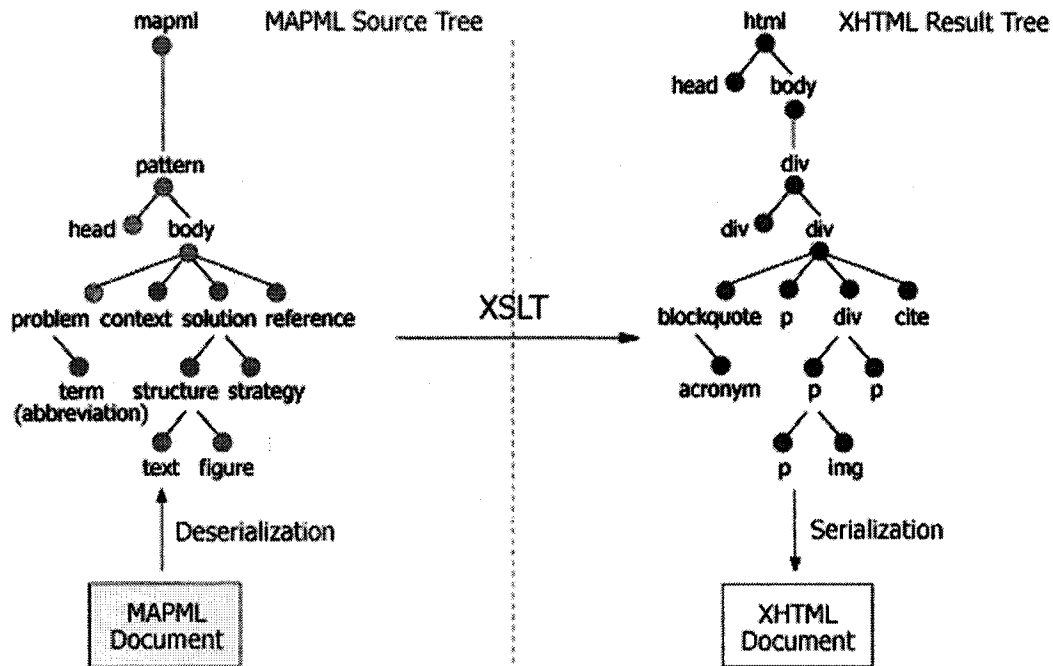


Figure 5.8. The MAPML to XHTML via XSLT Process.

5.4.3. Associating Style Sheets with MAPML-Related Documents

Once a style sheet for a MAPML has been authored, it needs to be processed in order for the rules in it to be applied to the MAPML document instance to obtain the final presentation. The same applies to an XHTML document instance resulting from a MAPML to XHTML Basic transformation. A style sheet processor needs a formalism for identifying the style sheet rules associated with the document. Therefore, XML-syntax based documents (and vocabularies, in general) need a standard way to associate style sheets.

5.4.3.1. XML Documents with Internal Style Sheets

Style information can be expressed within an XML document via internal style sheets. In case of XHTML, style sheet rules can be including in the document in the following form:

```
<style>
<!-- CSS Rules Here. -->
</style>
```

5.4.3.2. XML Documents with External Style Sheets

Style information can also be attached to an XML document via external style sheets. For example, in XHTML we can have:

```
<link rel="stylesheet" type="text/css" href="mobile.css"/>.
```

A general mechanism to associate a style sheet with an XML document is provided by the Associating Style Sheets with XML Documents Version 1.0 Specification [53]. The association consists of inserting the special XML processing instruction at the top of the document, before the root element of the XML document. The processing instruction has two required attributes, `type` and `href`, which respectively specify the style sheet type (Internet Media Type [34]) and its address (local directory path or the URI). An example is:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/css" href="nokia7110.css"?>
<mapml xmlns="http://www.cs.mcgill.ca/mapml">
  <pattern>
    <!-- Other Elements Here -->
  </pattern>
</mapml>
```

REMARKS

Neither the CSS style sheet nor the XSLT style sheet that we have created is unique. This is one significant advantage of the separation of structure and presentation.

The user has the freedom to create and associate their own style rules, complementing or replacing the ones that are already in existence.

In case of languages such as XHTML, style information can also be attached to elements via style attribute. For example:

```
<span style="background-color:
rgb(255,0,0)">Stop!</span>.
```

This practice mixes structure and presentation, and is discouraged.

5.5. Summary

This chapter presented the tools that are essential for working with MAPML: grammars that provide a formal description of the structure and syntax of MAPML and allow the validation of MAPML documents for conformance, authoring modes that enable syntax-sensitive editing, data binding code that facilitates the manipulation MAPML documents, and style sheets that enable the presentation of MAPML documents on the Web.

Chapter 6

Conclusion

Mobile applications are an important class of pervasive computing applications that provide the prospects of information and services available in multiple media formats, anytime, anywhere, in different modalities, and on a variety of devices with a diverse range of interfaces.

This thesis emphasizes a systematic approach of engineering mobile applications. Constructing these applications using XML under the direction provided by software engineering principles have various long-term benefits, both to the businesses and to the users. For these applications to achieve a high quality (be interoperable, be efficient, be usable, and so on), it is also important that use of open and standard technologies be made.

The patterns arising from the mobile application development can be classified into several semantically-separated categories: mobile business, mobile system architecture, mobile application process, mobile application architecture, mobile application product, and mobile application usage. The patterns in these classes can be represented using MAPML and interchanged in a device independent manner across a broad variety of devices and natural language environments. Using the utilities in MAPML-UTIL, MAPML documents can be authored on various platforms using simple editing modes, tested for conformance with respect to a variety of grammars, presented using style sheets on the Web in different ways, and processed in Java and C++.

There are several directions towards which MAPML and its supporting environment can be enhanced and/or improved:

- Chapter 5, Section 5.1 presented several ways of expressing a grammar for MAPML. The Schematron [31] is an XML Schema Language that is based on tree-patterns rather than regular-grammars such as XML DTD, XML Schema, and RELAX NG. Schematron can be implemented as XSLT style sheet. This makes Schematron processing widely available. There are several MAPML instances

where Schematron Schema would be useful. For example, if the `object-type` in the `object` element in MAPML is a `figure`, the `alternate` attribute should also be present. This **co-occurrence constraint** cannot be expressed in the grammar languages mentioned above, but it is possible to do that in Schematron. Therefore, the development of Schematron Schema for MAPML would be of interest.

- Limitations of current XML grammar languages towards data type validation were pointed out in Chapter 5, Section 5.1. Some of these can be circumvented by deploying a combination of techniques. This is the idea behind Document Schema Definition Language (DSDL) [29], a framework in progress under the auspices of ISO/IEC. Under DSDL, multiple validation tasks of different types can be applied to an XML document to achieve more complete validation results than just the application of a single technology. As the initiative matures, it would be of interest to examine how MAPML grammar support can be strengthened further.
- MAPML (and in general, XML) documents give rise to directed graphs (mostly trees) that can be modeled as objects. Artifacts of these object models and of those of associated grammars can be created using a visual modeling languages such as Unified Modeling Language (UML) and conceptual modeling techniques such as Object Role Model (ORM). There have been some (non-standard) early efforts in generating XML Schema Languages from these models [12]. It would be of interest to reverse-engineer MAPML XML Schema, express it in these models, and re-generate it. In that direction, the critical issue of the semantic transparency of the model-to-schema mapping is yet to be resolved.
- The thesis does not provide any new mobile application patterns. The work on creating the Mobile Application Patterns Catalog (MAPCAT), a catalog that consists of patterns for mobile devices with visual and speech interfaces, is in progress. Among the patterns that we have identified is the Single Source Transcoding Pattern where a single source is transformed to multiple formats for delivery and presentation to a range of different devices with a diverse range of capabilities. Figure 6.1 shows a schematic of this "author once, serve everywhere" approach. Figure 6.2 illustrates this in a specific case where documents based

upon book-oriented vocabularies can be transformed to the format of e-book vocabularies.

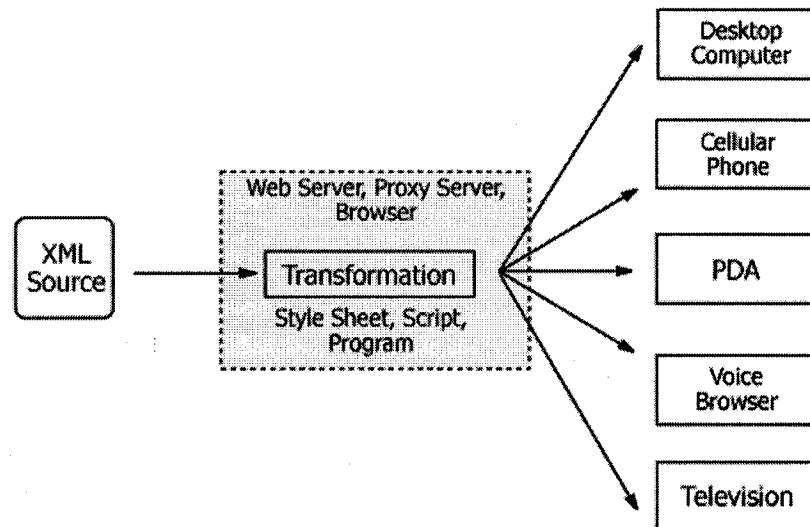


Figure 6.1. Role of XML Transformations in Delivering and Presenting [XML] Documents in Multiple-Environments.

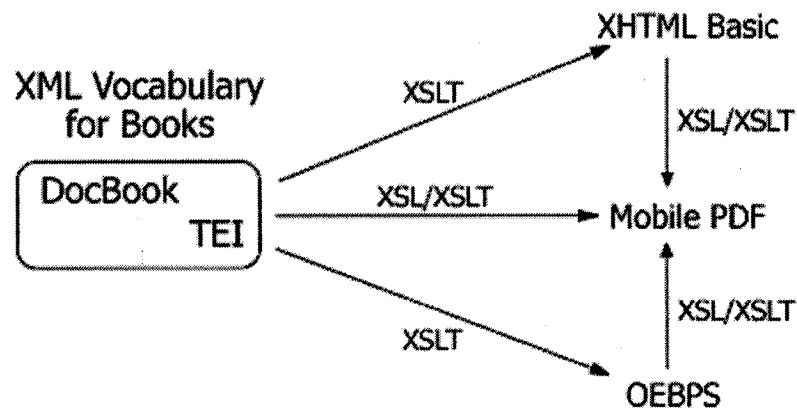


Figure 6.2. Transformations of XML Book Vocabularies to E-Book Vocabularies.

- Finally, it would also be useful to store patterns in MAPCAT in an XML database so that they can be readily submitted, automatically validated before storage, and precisely queried. We leave this work for future investigation.

Bibliography

- [1] Alexander, C. "The Timeless Way of Building", Oxford University Press, 1987.
- [2] Allamaraju, S. "Programming to XML - Data Binding Silver Bullet. Approaches and Alternatives", Presentation at XML 2001, Orlando, Florida, December 2001.
URL: <<http://www.idealliance.org/papers/xml2001/papers/html/04-01-03.html>>.
- [3] Alur, D., Crupi, J., Malks, D. "Pattern Template in Sun Java Center J2EE Patterns", Sun Java Center, March 2001.
URL: <<http://developer.java.sun.com/developer/technicalArticles/J2EE/patterns/PatternTemplate.html>>.
- [4] Appleton, B. Patterns and Software: Essential Concepts and Terminology". 2000.
URL: <<http://www.enteract.com/~bradapp/docs/patterns-intro.html>>.
- [5] Arciniegas, A. F. "Design Patterns in XML Applications, Part I: Traditional Patterns in XML applications", XML.com, January 2000.
URL: <<http://www.xml.com/pub/2000/01/19/feature/index.html>>.
- [6] Arciniegas, A. F. "Design Patterns in XML Applications, Part II: XML-specific patterns", XML.com, February 2000.
URL: <<http://www.xml.com/pub/2000/02/16/feature/index.html>>.
- [7] Berners-Lee, T., Fielding, R., Masinter, L. "RFC 2396: Uniform Resource Identifiers (URI): Generic Syntax", IETF, August 1998.
URL: <<http://www.ietf.org/rfc/rfc2396.txt>>.
- [8] Birbeck, M. et al. "Professional XML", Second Edition, Chapter 15: XML Data Binding, Wrox Press, 2001.
- [9] Bradner, S. "RFC 2119: Key words for use in RFCs to Indicate Requirement Levels", IETF, March 1997.
URL: <<http://www.ietf.org/rfc/rfc2119.txt>>.
- [10] Biggs, W. and Evans, H. "Simplify XML programming with JDOM", IBM developerWorks, May 2001.
URL: <<http://www.ibm.com/developerworks/java/library/j-jdom/>>.
- [11] Brownell, D. "SAX2", O'Reilly & Associates, January 2002.

- [12] Carlson, D. "Modeling XML Applications with UML", Addison-Wesley, 2001.
- [13] Crocker, D. H. "RFC 822: Standard for ARPA Internet Text Messages", IETF, August 1982.
URL: <<http://andrew2.andrew.cmu.edu/rfc/rfc822.html>>.
- [14] Durlacher Corporation Plc. Mobile Commerce Report - Durlacher Corporation Plc. Research Report, 1999.
URL: <<http://www.durlacher.com/research/res-reports.asp>>.
- [15] Freed, N. and Borenstein, N. "RFC 2045: Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", IETF, November 1996.
- [16] Gamma, E., Helm, R., Johnson, R., Vlissides, J. "Design Patterns -- Elements of Reusable Object-Oriented Software", Addison Wesley Professional Computing Series, 1994.
- [17] Graham, I., Quin, L. "Introduction to XML Design Patterns", 1999.
URL: <<http://www.grovieware.com/>>.
- [18] Griffiths, R., Pemberton, L. "Don't Write Guidelines, Write Patterns", University of Brighton, Brighton, UK, 2001.
URL: <<http://www.it.bton.ac.uk/staff/lp22/guidelinesdraft.html>>.
- [19] Guenette, D., Trippe, B. "E-books: Technology for Enterprise Content Applications?", The Gilbane Report on Open Information & Document Systems, Volume 8, Number 9, November 2000.
- [20] IBM. "Patterns for E-Business", IBM, 1999.
URL: <<http://www.ibm.com/developerworks/patterns/>>.
- [21] IBM. "Xena", IBM alphaWorks, March 1999.
URL: <<http://www.alphaworks.ibm.com/tech/xena>>.
- [22] IBM. "XML Generator", IBM alphaWorks, September 1999.
URL: <<http://www.alphaworks.ibm.com/tech/xmlgenerator>>.
- [23] IBM. "IBM Ease-of-Use > Web Design Guidelines > Design", IBM, July 2001.
URL: <http://www-3.ibm.com/ibm/easy/eou_ext.nsf/Publish/574>.
- [24] IBM. "IBM Web Design Guidelines", IBM, July 2001.
URL: <http://www-3.ibm.com/ibm/easy/eou_ext.nsf/Publish/572>.

- [25] IEEE, "Recommended Practice for Software Design Descriptions", IEEE STD 1016-1987, 1987.
- [26] IEEE, "Recommended Practice for Software Requirements Specifications", IEEE STD 830-1998, 1998.
- [27] ISO/IEC. "ISO/IEC 14977:1996. Information technology - Syntactic metalanguage - Extended BNF", ISO/IEC, 1996.
- [28] ISO (International Organization for Standardization). "ISO/IEC 10646-1:2000. Information technology -- Universal Multiple-Octet Coded Character Set (UCS) -- Part 1: Architecture and Basic Multilingual Plane", International Organization for Standardization, 2000.
- [29] Bryan, M. (Project Editor). "ISO/IEC JTC 1/SC34 Information Technology -- Document Description and Processing Languages", Working Draft, October 22, 2001.
- [30] Jacobson, I., Christerson, M., Jonsson P., Övergaard, G. "Object-Oriented Software Engineering: A Use Case Driven Approach", Addison-Wesley, 1992.
- [31] Jelliffe, R. "The Schematron Assertion Language", Academia Sinica Computing Centre. October 2000.
- URL: <<http://www.ascc.net/xml/resource/schematron/>>.
- [32] Kamthan, Pankaj. "XML Characterization Project", September 2001.
- [33] Knuth, Donald E. "Literate Programming", CSLI Lecture Notes, Number 27, 1992.
- [34] Kohn, D., Murata, M., St. Laurent, S. "RFC 3023: XML Media Types", IETF, January 2001.
- URL: <<http://www.ietf.org/rfc/rfc3023.txt>>.
- [35] Kuhn, Markus. "A Summary of the International Standard Date and Time Notation", University of Cambridge, November 2001.
- URL: <<http://www.cl.cam.ac.uk/~mgk25/iso-time.html>>.
- [36] Lainevoel, Toivo. "XML Patterns", XMLPatterns.com, 2000.
- URL: <<http://www.xmlpatterns.com/>>.
- [37] Lea, D. Patterns-Discussion FAQ - By Doug Lea . November 2000.
- URL: <<http://g.oswego.edu/dl/pd-FAQ/pd-FAQ.html>>.
- [38] Lee, D., Chu, W. W. "Comparative Analysis of Six Schema Languages", University of California, Los Angeles, August 2000.

- URL: <<http://www.cobase.cs.ucla.edu/tech-docs/dongwon/ucla-200008.html>>.
- [39] Li, Ning. "Usability Patterns-Assisted Design for Web User Interfaces", Masters Thesis, Concordia University, Canada, October 2001.
- [40] Lord, John. "Facilitating the application development process using the IBM Patterns for e-business", IBM, August 2001.
- URL: <<http://www-106.ibm.com/developerworks/patterns/guidelines/lord.pdf>>.
- [41] Clark, J., Murata, M. (Editors) "RELAX NG Specification", OASIS Committee Specification, December 2001.
- URL: <<http://www.oasis-open.org/committees/relax-ng/spec-20011203.html>>.
- [42] Object Management Group (OMG). "Unified Modeling Language, Version 1.4 Specification", Object Management Group, Inc. 2001.
- [43] Powell, A. and Pete Johnston, P. "Guidelines for implementing Dublin Core in XML", UKOLN, University of Bath. January 2002.
- URL: <<http://www.ukoln.ac.uk/metadata/dcmi/dc-xml-guidelines/>>.
- [44] Rosenfeld, L., Morville, P. "Information Architecture for the World Wide Web. Designing Large-Scale Web Sites", O'Reilly & Associates, Inc., 1998.
- [45] The Unicode Consortium. "The Unicode Standard, Version 3.0". Addison-Wesley, 2000.
- [46] Tomihisa Kamada, T. (Author). "Compact HTML for Small Information Appliances", W3C Note, February 9, 1998.
- URL: <<http://www.w3.org/TR/1998/NOTE-compactHTML-19980209>>.
- [47] Bos, B., Lie, H-W., Lilley C., Jacobs, I. (Editors). "Cascading Style Sheets, Level 2 (CSS2) Specification", W3C Recommendation, May 1998.
- URL: <http://www.w3.org/TR/REC-CSS2>.
- [48] Hjelm, J., King, P., Martin, B. (Authors). "WAP Forum - W3C Cooperation White Paper", W3C Note, October 1998.
- URL: <<http://www.w3.org/TR/NOTE-WAP>>.
- [49] Bray, T., Hollander, D., Layman, A. (Editors). "Namespaces in XML", W3C Recommendation, January 1999.
- URL: <<http://www.w3.org/TR/REC-xml-names>>.

- [50] Chisholm, W., Vanderheiden, G., Jacobs, I. (Editors). "Web Content Accessibility Guidelines 1.0 - W3C Recommendation", W3C Recommendation, May 1999.
URL: <<http://www.w3.org/TR/WAI-WEBCONTENT>>.
- [51] Dawkins, S., Hjelm, J., Reynolds, F., Singhal, S. (Editors). "Composite Capability/Preference Profiles (CC/PP): a user-side framework for content negotiation", W3C Note, July 1999.
URL: <<http://www.w3.org/TR/NOTE-CCPP>>.
- [52] Clark, J. (Editor). "XSL Transformations (XSLT) Version 1.0", W3C Recommendation, November 1999.
URL: <<http://www.w3.org/TR/xslt>>.
- [53] Clark, J. (Editor). "Associating Style Sheets with XML Documents Version 1.0", W3C Recommendation, June 1999.
URL: <<http://www.w3.org/TR/xml-stylesheet>>.
- [54] Hors, A., Jacobs, I., Raggett, D. (Editors). "HTML 4.01 Specification - W3C Recommendation", W3C Recommendation, December 1999.
URL: <<http://www.w3.org/TR/html401/>>.
- [55] Bray, T., Paoli, J., Sperberg-McQueen, C. M., Maler, E. (Editors). "Extensible Markup Language (XML) 1.0 (Second Edition)", W3C Recommendation, October 2000.
URL: <<http://www.w3.org/TR/REC-xml>>.
- [56] Davis, M., Hors, A., Robie, J., Wood, L. et al. (Editors). "Document Object Model (DOM) Level 2 Core Specification", W3C Recommendation, November 2000.
URL: <<http://www.w3.org/TR/DOM-Level-2-Core/>>.
- [57] Baker, M., Ishikawa, M., Matsui, S. et al. (Editors). "XHTML Basic", W3C Recommendation, December 2000.
URL: <<http://www.w3.org/TR/xhtml-basic>>.
- [58] Altheim, M., Boumphrey, F., Dooley, S., McCarron, S., Schnitzenbaumer, S., Wugofski, T. (Editors). "Modularization of XHTML™", W3C Recommendation, May 2001.
URL: <<http://www.w3.org/TR/xhtml-modularization>>.
- [59] Fallside, David C. (Editor). "XML Schema Part 0: Primer", W3C Recommendation, May 2001.

URL: <<http://www.w3.org/TR/xmlschema-0>>.

[60] DeRose, S., Maler, E., Orchard, E., (Editors). "XML Linking Language (XLink) Version 1.0 - W3C Recommendation", W3C Recommendation, June 2001.

URL: <<http://www.w3.org/TR/xlink>>.

[61] Gimson, Roger. (Editor). "Device Independence Principles", W3C Working Draft, September 2001.

URL: <<http://www.w3.org/TR/di-princ/>>.

[62] Dardailler, D., Palmer, S. B. (Editors). "XML Accessibility Guidelines", W3C Working Draft, August 2001.

URL: <<http://www.w3.org/TR/xmlgl>>.

[63] Altheim, M., McCarron, S. (Editors). "XHTML™ 1.1 - Module-based XHTML", W3C Recommendation, May 2001.

URL: <<http://www.w3.org/TR/xhtml11>>.

[64] W3C, WAP Forum. "Position Dependent Information Services", W3C-WAP Forum Workshop, February 2000.

URL: <<http://www.w3.org/Mobile/posdep/>>.

[65] Waldhoff, Rod. "Towards a DTD for Documenting Patterns and Pattern Languages in XML", June 1998.

URL: <http://members.tripod.com/~rwald/patterns/pat_dtd.html>.

[66] Wireless Application Group. "User Agent Profile Specification", WAP Forum, November 1999.

URL: <<http://www.wapforum.org/what/technical.htm>>.

[67] Wireless Application Group. "Wireless Markup Language (WML). Version 2.0.", WAP Forum, January 2002.

URL: <<http://www.wapforum.org/what/technical.htm>>.

[68] Welie, M. V. "Web Design Patterns", Welie.com, 2001.

URL: <<http://www.welie.com/patterns/>>.

Appendix A

MAPML Schema

```
<?xml version="1.0" encoding="UTF-8"?>

<schema
  xmlns="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.cs.mcgill.ca/mapml"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:mapml="http://www.cs.mcgill.ca/mapml"
  elementFormDefault="qualified"
  attributeFormDefault="qualified">
  <!-- ..... -->
  <annotation>
    <documentation xml:lang="en">

      File: mapml.xsd
      Author: Hsueh-Ieng Pai
      Date: December 15, 2001
      Version: 1.0
      Description: XML Schema for Mobile Application Patterns Markup
                   Language (MAPML).
                   This XML Schema is identified by the XML Namespace:
                   http://www.cs.mcgill.ca/mapml
      Copyright (c) 2001 Hsueh-Ieng Pai.
      Permission is granted to copy, distribute and/or modify this
      Document under the terms of the GNU Free Documentation License,
      Version 1.1 or any later version published by the Free Software
      Foundation.

    </documentation>
  </annotation>
  <!-- ..... -->
  <!-- Import xml:lang and xml:space Attributes ..... -->
  <import namespace="http://www.w3.org/XML/1998/namespace"
    schemaLocation="http://www.w3.org/2001/xml.xsd">
  </import>

  <!-- ..... -->
  <!-- ASSOCIATION MODULE ..... -->
  <!-- ..... -->

  <!-- The class Element ..... -->
  <element name="class">
    <complexType>
      <choice>
        <element ref="mapml:term" minOccurs="0" maxOccurs="unbounded"/>
      </choice>
      <attribute name="id" type="ID" use="optional"/>
    </complexType>
  </element>
```

```

<!-- The link Element ..... -->
<element name="link">
  <complexType>
    <simpleContent>
      <extension base="string">
        <attribute name="id" type="ID" use="optional"/>
        <attribute name="uri" type="anyURI" use="required"/>
      </extension>
    </simpleContent>
  </complexType>
</element>

<!-- The pattern.related Element ..... -->
<element name="pattern.related">
  <complexType mixed="true">
    <choice maxOccurs="unbounded">
      <element ref="mapml:name"/>
      <element ref="mapml:class"/>
      <element ref="mapml:link"/>
    </choice>
    <attribute name="id" type="ID" use="optional"/>
    <attribute name="relation" use="required">
      <simpleType>
        <restriction base="string">
          <enumeration value="superordinate"/>
          <enumeration value="subordinate"/>
          <enumeration value="sibling"/>
          <enumeration value="competitor"/>
        </restriction>
      </simpleType>
    </attribute>
  </complexType>
</element>

<!-- The reference Element ..... -->
<element name="reference">
  <complexType>
    <choice>
      <element ref="mapml:title"/>
      <element ref="mapml:author" minOccurs="1" maxOccurs="unbounded"/>
      <element ref="mapml:date" minOccurs="1" maxOccurs="unbounded"/>
      <element ref="mapml:link"/>
    </choice>
    <attribute name="id" type="ID" use="required"/>
  </complexType>
</element>

<!-- ..... -->
<!-- META-INFORMATION MODULE ..... -->
<!-- ..... -->

<!-- The abstract Element ..... -->
<element name="abstract">
  <complexType mixed="true">
    <choice>
      <element ref="mapml:term" minOccurs="0" maxOccurs="unbounded"/>
    </choice>
  </complexType>
</element>

```



```

    <attribute name="id" type="ID" use="optional"/>
  </complexType>
</element>

<!-- The author Element ..... -->
<element name="author">
  <complexType>
    <attribute name="id" type="ID" use="optional"/>
  </complexType>
</element>

<!-- The caption Element ..... -->
<element name="caption">
  <complexType mixed="true">
    <choice>
      <element ref="mapml:term" minOccurs="0" maxOccurs="unbounded"/>
    </choice>
  </complexType>
</element>

<!-- The date Element ..... -->
<element name="date">
  <complexType>
    <simpleContent>
      <extension base="string">
        <attribute name="id" type="ID" use="optional"/>
      </extension>
    </simpleContent>
    <attribute name="event" use="optional">
      <simpleType>
        <restriction base="string">
          <enumeration value="creation"/>
          <enumeration value="publication"/>
          <enumeration value="revision"/>
        </restriction>
      </simpleType>
    </attribute>
  </complexType>
</element>

<!-- The description Element ..... -->
<element name="description">
  <complexType mixed="true">
    <choice>
      <element ref="mapml:term" minOccurs="0" maxOccurs="unbounded"/>
    </choice>
    <attribute name="id" type="ID" use="optional"/>
  </complexType>
</element>

<!-- The keyword Element ..... -->
<element name="keyword">
  <complexType mixed="true">
    <choice>
      <element ref="mapml:term" minOccurs="1" maxOccurs="unbounded"/>
    </choice>
  </complexType>

```



```

</element>

<!-- The license Element ..... -->
<element name="license">
  <complexType mixed="true">
    <choice>
      <element ref="mapml:author" minOccurs="1" maxOccurs="unbounded"/>
      <element ref="mapml:date" minOccurs="1" maxOccurs="unbounded"/>
    </choice>
    <attribute name="id" type="ID" use="optional"/>
  </complexType>
</element>

<!-- The metadata Element ..... -->
<element name="metadata">
  <complexType>
    <sequence>
      <element ref="mapml:name"/>
      <element ref="mapml:class"/>
      <element ref="mapml:author" minOccurs="1" maxOccurs="unbounded"/>
      <element ref="mapml:date" minOccurs="0" maxOccurs="unbounded"/>
      <element ref="mapml:abstract"/>
      <element ref="mapml:license" minOccurs="0" maxOccurs="1"/>
      <element ref="mapml:keyword"/>
    </sequence>
  </complexType>
</element>

<!-- The name Element ..... -->
<element name="name">
  <complexType>
    <simpleContent>
      <extension base="string">
        <attribute name="id" type="ID" use="optional"/>
      </extension>
    </simpleContent>
  </complexType>
</element>

<!-- The term Element ..... -->
<element name="term">
  <complexType>
    <attribute name="term-type" use="required">
      <simpleType>
        <restriction base="string">
          <enumeration value="abbreviation"/>
          <enumeration value="concept"/>
          <enumeration value="principle"/>
          <enumeration value="technology"/>
          <enumeration value="tool"/>
        </restriction>
      </simpleType>
    </attribute>
  </complexType>
</element>

```

```

<!-- The title Element ..... -->
<element name="title" type="string">
  </element>

<!-- ..... -->
<!-- PROBLEM MODULE ..... -->
<!-- ..... -->

<!-- The constraint Element ..... -->
<element name="constraint">
  <complexType mixed="true">
    <choice>
      <element ref="mapml:term" minOccurs="0" maxOccurs="unbounded"/>
    </choice>
    <attribute name="id" type="ID" use="optional"/>
  </complexType>
</element>

<!-- The context Element ..... -->
<element name="context">
  <complexType mixed="true">
    <choice>
      <element ref="mapml:term" minOccurs="0" maxOccurs="unbounded"/>
    </choice>
    <attribute name="id" type="ID" use="optional"/>
  </complexType>
</element>

<!-- The problem Element ..... -->
<element name="problem">
  <complexType mixed="true">
    <choice>
      <element ref="mapml:term" minOccurs="0" maxOccurs="unbounded"/>
      <element ref="mapml:reference" minOccurs="0"
        maxOccurs="unbounded"/>
    </choice>
    <attribute name="id" type="ID" use="optional"/>
  </complexType>
</element>

<!-- ..... -->
<!-- SOLUTION MODULE ..... -->
<!-- ..... -->

<!-- The consequence Element ..... -->
<element name="consequence">
  <complexType mixed="true">
    <choice>
      <element ref="mapml:term" minOccurs="0" maxOccurs="unbounded"/>
      <element ref="mapml:reference" minOccurs="0"
        maxOccurs="unbounded"/>
    </choice>
    <attribute name="id" type="ID" use="optional"/>
    <attribute name="impact" use="required">
      <simpleType>
        <restriction base="boolean">
          <enumeration value="positive"/>

```

```

        <enumeration value="negative"/>
    </restriction>
</simpleType>
</attribute>
</complexType>
</element>

<!-- The implementation Element ..... -->
<element name="implementation">
    <complexType mixed="true">
        <choice>
            <element ref="mapml:object" minOccurs="0" maxOccurs="unbounded"/>
            <element ref="mapml:term" minOccurs="0" maxOccurs="unbounded"/>
        </choice>
        <attribute name="id" type="ID" use="optional"/>
    </complexType>
</element>

<!-- The object Element ..... -->
<element name="object">
    <complexType>
        <sequence>
            <element ref="mapml:caption"/>
            <element ref="mapml:link"/>
        </sequence>
        <attribute name="id" type="ID" use="optional"/>
        <attribute name="content-type" type="string" use="required"/>
        <attribute name="object-type" use="required">
            <simpleType>
                <restriction base="string">
                    <enumeration value="figure"/>
                    <enumeration value="code"/>
                    <enumeration value="markup"/>
                </restriction>
            </simpleType>
        </attribute>
        <attribute name="alternate" type="string" use="optional"/>
    </complexType>
</element>

<!-- The rationale Element ..... -->
<element name="rationale">
    <complexType mixed="true">
        <choice>
            <element ref="mapml:term" minOccurs="0" maxOccurs="unbounded"/>
        </choice>
        <attribute name="id" type="ID" use="optional"/>
    </complexType>
</element>

<!-- The scenario Element ..... -->
<element name="scenario">
    <complexType mixed="true">
        <choice>
            <element ref="mapml:object" minOccurs="0" maxOccurs="unbounded"/>
            <element ref="mapml:term" minOccurs="0" maxOccurs="unbounded"/>
        </choice>
    </complexType>
</element>

```



```

    <attribute name="id" type="ID" use="optional"/>
  </complexType>
</element>

<!-- The solution Element ..... -->
<element name="solution">
  <complexType mixed="true">
    <sequence>
      <element ref="mapml:structure"/>
      <element ref="mapml:strategy"/>
    </sequence>
    <choice>
      <element ref="mapml:term" minOccurs="0" maxOccurs="unbounded"/>
    </choice>
    <attribute name="id" type="ID" use="optional"/>
  </complexType>
</element>

<!-- The strategy Element ..... -->
<element name="strategy">
  <complexType mixed="true">
    <choice>
      <element ref="mapml:implementation" minOccurs="1"
        maxOccurs="unbounded"/>
      <element ref="mapml:term" minOccurs="0" maxOccurs="unbounded"/>
    </choice>
    <attribute name="id" type="ID" use="optional"/>
  </complexType>
</element>

<!-- The structure Element ..... -->
<element name="structure">
  <complexType mixed="true">
    <choice>
      <element ref="mapml:object" minOccurs="0" maxOccurs="unbounded"/>
      <element ref="mapml:term" minOccurs="0" maxOccurs="unbounded"/>
    </choice>
    <attribute name="id" type="ID" use="optional"/>
  </complexType>
</element>

<!-- ..... -->
<!-- STRUCTURE MODULE ..... -->
<!-- ..... -->

<!-- The body Element ..... -->
<element name="body">
  <complexType mixed="true">
    <sequence>
      <element ref="mapml:description"/>
      <element ref="mapml:context"/>
      <element ref="mapml:problem"/>
      <element ref="mapml:constraint" minOccurs="1"
        maxOccurs="unbounded"/>
      <element ref="mapml:rationale"/>
      <element ref="mapml:solution"/>
      <element ref="mapml:consequence" minOccurs="0"

```

```

        maxOccurs="unbounded"/>
    <element ref="mapml:pattern.related" minOccurs="1"
        maxOccurs="unbounded"/>
    <element ref="mapml:scenario" minOccurs="1"
        maxOccurs="unbounded"/>
</sequence>
</complexType>
</element>

<!-- The head Element ..... -->
<element name="head">
    <complexType mixed="true">
        <sequence>
            <element ref="mapml:metadata"/>
        </sequence>
    </complexType>
</element>

<!-- The mapml Element ..... -->
<element name="mapml">
    <complexType>
        <sequence>
            <element ref="mapml:pattern" minOccurs="1" maxOccurs="unbounded"/>
        </sequence>
        <attribute name="id" type="ID" use="optional"/>
        <attribute name="version" type="decimal" fixed="1.0"/>
        <attribute ref="xml:lang" use="required"/>
        <attribute ref="xml:space" default="preserve"/>
    </complexType>
</element>

<!-- The pattern Element ..... -->
<element name="pattern">
    <complexType>
        <sequence>
            <element ref="mapml:head"/>
            <element ref="mapml:body"/>
        </sequence>
        <attribute name="id" type="ID" use="optional"/>
        <attribute name="version" type="decimal"/>
    </complexType>
</element>
</schema>

<!-- ..... -->

```