

# Writing Use Cases in XML

Dimitris Dranidis, Kalliopi Tigka

Computer Science Department, CITY Liberal Studies  
Affiliated College of the University of Sheffield  
13 Tsimiski str., 54624 Thessaloniki, Greece  
Email: {dranidis,tigka}@city.academic.gr

**Abstract.** Writing use cases is the most important step for modelling user requirements. Use cases focus on describing scenarios of using the system and are usually written in a textual form. To assist use case writing, various templates have been proposed, mainly differing in their visual representation. XML is a platform independent language for describing data and their structure without referring to their visual presentation. In this paper, we propose the writing of use cases in XML. Using XML, we achieve the clear separation of the semantic part of use case descriptions from their visual representations. Hence, the same description can be visualized in many different ways. Furthermore, our encoding allows the integration of tools for writing and validating use cases. We introduce an appropriate structure for use case descriptions and we model this structure using DTD (Document Type Definition). We demonstrate the applicability of our approach, by presenting an example of a use case description and its transformation in multiple representation formats.

## 1 Introduction

Use cases constitute the basic way of modelling user requirements during software development. Nevertheless, there exist no standard rules for writing them. During the last years, many templates have been proposed for writing use cases [1, 3, 9]. Some examples are the fully-dressed, the table, and the two-column format. Essentially, all these formats contain the same information presented in a different way. Also, different levels of detail are needed in different phases of software development. This need presupposes that the information is not degraded during its transformation in more detailed forms.

XML (eXtensible Markup Language) [10] is a language for data description, which has the potential of representing information in different forms, as well as, in different levels of detail. Compared to other documents, such as HTML and PDF, an XML document is not bound to an explicit output format. With the description written in XML, it is possible to produce any of the desired formats without rewriting the contents.

Our aim is to define a technique that will assist developers in writing use cases, without bothering about its visual representation, while at the same time respecting the logical structure of a use case. For visualization purposes, we provide several XSLT transformations that can be applied and result different visual formats in different levels of detail.

This paper is organized as follows: In the next section, we discuss about use cases and the different types of their presentation. Section 3 briefly presents XML and XSLT. In section 4, we model use cases in XML and demonstrate our approach by producing different types of visual data representation using an example. Finally, we close the report with some conclusions in section 5.

## 2 Use Cases

Use cases were introduced by Jacobson in 1987 [5] as a technique for documenting functional requirements of a system under development. Ten years later, the Unified Modelling Language (UML) [7] has adopted this technique. Nowadays, the use case approach is widely used for describing user requirements.

User requirements are modelled twofold: writing use cases descriptions and constructing a use case diagram. A use case diagram is composed of use cases and represents the global system's behavior. Each use case corresponds to an exact behavior of the system that satisfies a user goal. Formally, a use case is defined as a sequence of events that illustrates the behavior of a system and its components when a service is invoked [6]. This behavior is described in several ways but the most common one is the textual description.

Although use case diagrams and use cases are practical, they have important semantic weaknesses in their notation. The use case description is usually written in simple text. There are no official guidelines for writing use cases, yet several researchers [1–3] propose different techniques and practices to describe use case scenarios. We distinguish three basic presentation forms of use cases:

**Natural Language Text** Simple textual description is the easiest and the most popular way for specifying a use case. This is an informal specification and thus includes many ambiguities. In a high level of analysis, the textual description is useful because it gives freedom of expression.

**Tabular Representations** This presentation is based on structured and sequential textual description. The progression of rows in the table encodes the flow of the events. Events are numbered and each scenario is described as a unique sequence of events. In a two-column representation, entries in the first column correspond to the actors' interactions, while entries in the second column are descriptions of the system actions. In a single column vertical sequence, a total ordering of the events is represented by the rows.

**Directed Graphs** Graphs are used for the simultaneous representation of all the scenarios of a use case. A node corresponds to a state of the system and an arc to a transition fired by an event.

Use cases are central artifacts in many phases of the software development cycle. During the requirement analysis, they are useful to specify requirements. At the design phase, they drive the creation of interaction diagrams. During the testing of the system, they are used for deriving test cases.

The utilization of use cases in different phases of the software life cycle implies different levels of precision. Simple textual descriptions are sufficient during the analysis

since no information concerning the internal behavior of the system is known. In the design phase, more detailed information is presented with tabular representations in which the flow of events is clearly shown. Directed graphs, on the other hand, can help to the creation of the corresponding test cases. Consequently, a single presentation of a use case is not sufficient.

During development, several representations of the same use case, constructed incrementally, should be maintained. Furthermore, a use case refinement should not change the behavior given in the main description. Thus, a technique is required for the structured description of use cases, which allows the incremental addition of detail, by preserving the properties of the original. The technique should allow multiple representations of the same use case, via transformations. As we demonstrate in this paper, our approach achieves these objectives.

### 3 XML

XML is an extensible mark-up language that was published as a W3C recommendation in 1998 [10]. Some of the objectives for the design of XML were the easy usage over the Internet, the legibility of the documents, and the minimality of the language [10]. XML supports a wide variety of applications such as logical data description (databases, documents) [4, 8] and conceptual data description [3]. Extensibility is a key feature of XML. Authors can declare and use their own tags and attributes. XML offers a convenient syntax for data representation providing little semantic information. Thus, XML emphasizes on the description of the content separately from its presentation.

Although an XML description already implies a logical data structure, in order to validate this structure, syntax rules may be provided explicitly. XML offers the possibility of creating document type definitions (DTD) that define constraints on the logical structure of XML documents. A DTD defines a grammar of element types and attributes for each element. An XML document that conforms to an associated DTD structure is called “valid”.

XML does not deal with presentation issues; presentation is addressed by XSLT (XML Style Language Transformations). XSLT [11] is a rule-based declarative language, which is used for writing transformations. An XSLT transformation parses a valid and well-formed XML document and transforms it into another document. The transformation might range from a simple visualization of the contained data (e.g. in HTML), to a complex processing or restructuring of the data for any desired purpose. XSLT was designed for use as part of XSL (extensible stylesheet language for XML). XSL [12] also includes an XML vocabulary for specifying formatting. XSLT is used to describe how the document is transformed into another XML document that uses the formatting vocabulary. Nevertheless, XSLT was also designed to be used independently of XSL, as we do in this paper. The transformations that we produce are mainly to HTML documents for previewing.

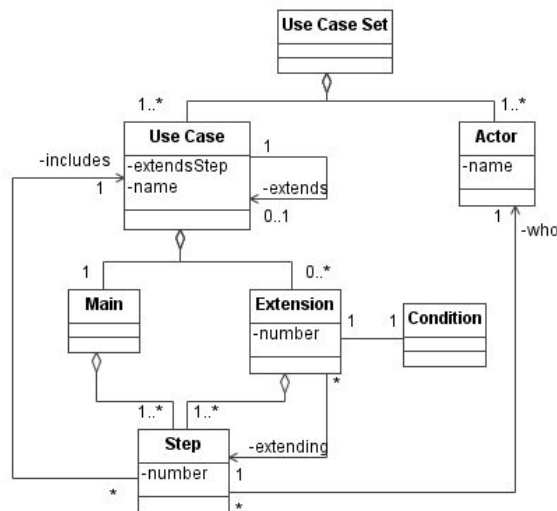
## 4 Modelling Use Cases in XML

To model use cases in XML, we have to define the logical structure that the description should follow. By analyzing the concepts related to use cases, we construct a logical meta-model of use cases and present this using a UML class diagram. Then we transform the class diagram into a document type definition (DTD) for XML documents. This DTD file presents the structure and the validity constraints of XML documents modelling use cases. Using the DTD one may write use cases in XML and validate them. Although the descriptions written in XML are fairly legible, the presence of tags may confuse readers. Therefore, we define XSLT transformations that present use case descriptions in several common formats.

### 4.1 Modelling the Use Case Structure in UML

In this section, a standard structure for use cases is presented. This structure is semiformal and contains the most important information for describing use cases. We provide a UML class diagram (Figure 1) to illustrate the structure.

The functional requirements of a system are described as a set of use cases (use case set). The use case set is defined as a composition of all actors (users of the system) and use cases (the distinct functionalities that the system provides to its users).



**Fig. 1.** Class diagram showing the structure of a use case set.

A use case may consist of several scenarios. One of them is the main scenario and the rest are called alternative scenarios. A scenario describes a flow of events. One of the actors causes the first event that activates the use case. The rest of the events

are caused by a system component or by an actor, when the system needs more information to go on. The main scenario describes the most usual flow of actor-system interactions. It corresponds to the case in which no problems occur. The alternative scenarios describe the behavior of the system when exceptional events happen. Alternative scenarios are usually modelled by extensions. An extension is a sequence of extra steps that is activated when a condition is satisfied (the condition corresponds to the occurrence of an exceptional event).

In the depicted class diagram, a use case is defined as a composition of exactly one main scenario and any number of extensions. Further, both the main scenario and the extensions are composed by several steps that are distinguished by a unique number and they are activated by the actor or from the system itself (role name *who* of the association). An extension is activated when a condition is satisfied and refers to an exact step (role name *extending* of the association) of the main scenario.

UML defines three types of relationships between use cases: “include”, “extend”, and “generalize”. The most frequently used is the “include” relationship. Parts of scenarios that are repeated in many use cases are usually described separately and then included in any step of a use case by simply referring to the included use case. The “extend” relationship is not used so frequently. In cases in which extensions are too lengthy or complicated, it is advised to model them as separate use cases on their own. The extension use case then is said to extend the main use case at a specific step under some condition. Finally, the generalization relationship is used rarely; we do not deal with this relationship in our modelling. The “include” relationship is defined as a role name of a use case associated with a step. Any step may include a use case; it is common practice to include the same use case in many steps in different use cases (this is the usefulness of “include”). The “extend” relationship is defined as a role name of the use case that extends the main use case. The attribute *extendsStep* defines the step of the main use case that is being extended.

## 4.2 Modelling the Use Case Structure in a DTD Description

By translating the class diagram presented, we create a DTD description (Figure 2). Each of the classes corresponds to an entry element. The composition relationship is modelled as a containment relationship; for instance, the element `UseCaseSet` contains as children (at least one) `actor` and (at least one) `usecase` elements. Role names are modelled as attributes in the attribute list of the elements; for instance, `extends` is defined as an optional attribute of element `usecase`.

The validity constraints in the DTD guarantee a simple kind of referential integrity of the XML document contents. Values of type ID cannot appear more than once in an XML document as a value of this type. This guarantees, for instance, that all use cases carry unique names; the same holds for actors. IDREF values must match the value of some ID attribute. For instance, an included use case name in a step (`includes` attribute of element `step`) must already exist as the name of one of the use cases.

```

<!ELEMENT UseCaseSet (actor+, usecase+)>
<!ELEMENT actor EMPTY>
<!ATTLIST actor name ID #REQUIRED>
<!ELEMENT usecase (main, extension*)>
<!ATTLIST usecase
  name ID #REQUIRED
  extends IDREF #IMPLIED
  extendsStep CDATA #IMPLIED>
<!ELEMENT main (step+)>
<!ELEMENT extension (condition, step+)>
<!ATTLIST extension
  extending CDATA #REQUIRED
  number CDATA #REQUIRED>
<!ELEMENT step (#PCDATA)>
<!ATTLIST step
  number CDATA #REQUIRED
  who IDREF #REQUIRED
  includes IDREF #IMPLIED>
<!ELEMENT condition (#PCDATA)>

```

Fig. 2. Document Type Definition (DTD) for the use cases.

### 4.3 XSLT Transformations

In this section, we describe some of the possible XSLT transformations that could be of value for developers. We separate transformations in four categories: visualization, validation, metrics, and tools integration.

**Visualization** An XML document can be transformed into and viewed as an HTML file (or any other output form such as PDF,  $\text{\TeX}$ , and RTF) using the appropriate XSLT transformation. Most important than that, the format of the use case can be in any of the common template formats, such as the simple text, one-column, and two-column format. This solves the problem of choosing the appropriate format before writing the use cases. Decisions to change the representations do not affect the written use cases, since they are written independently of any visualization. In particular, the transformation into HTML allows the automatic integration of hyperlinks in the document. This results in a hypertext software documentation, which is easily accessible, navigable, and distributable in an enterprises network.

**Validation** The validation of an XML document against its associated DTD already provides a means of checking use cases for consistency. The validity constraints of the DTD guarantee that: (a) all actors initiating use case steps are specified beforehand as members of the use case set; (b) included and extended use cases refer only to existing use cases. Apart from these checks, we may define XSLT transformations that perform more complex tasks.

**Metrics** Interesting measurements can be easily derived by suitable XSLT transformations such as the number of use cases and actors, and the average number of

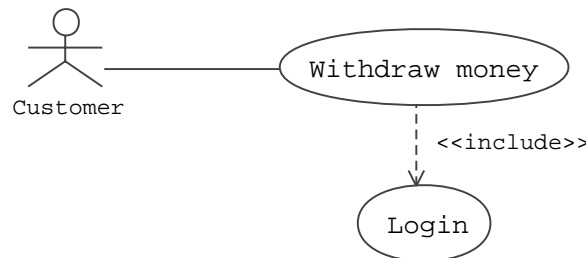
steps or extensions. When regularly collected, these measurements can be used for the assessment of the quality, the estimation of the complexity of the project, and further project estimation purposes.

**Tools integration** One of the objectives of XML is the easy exchange of information between many applications. Transformations may be written that transform the use case documents into formats readable by existing CASE tools. Apart from that, the collections of all XSLT transformations (for visualization, validation, metrics, etc.) can play the role of an open and extensible repository of tools for requirements engineering. An application providing a graphical use interface for writing the XML files and offering all transformations will be of a great value to developers.

In the next section we demonstrate only the application of visualization transformations.

#### 4.4 An Example: ATM

Consider an Automatic Teller Machine (ATM), which works outside of a bank. Every time a customer inserts a magnetic card, the machine validates the data given from the customer and then performs the selected transaction. There are several different transactions that may be performed such as withdraw money, deposit money, and request of balance information. Each one of them corresponds to a use case and includes more than one scenario. Figure 3 illustrates a partial use case diagram for the ATM, showing the use case “Withdraw money” and the included use case “Login”.



**Fig. 3.** Use case diagram.

The use case “Login” describes the flow of events that refers to the login process. This process is activated only once in the beginning of the first interaction with the ATM (e.g. deposit money or get balance information). Suppose that the following are the descriptions of the use cases as provided by users of the ATM. Both descriptions are written in a relatively free “natural language text” format.

**Use case:** Login  
**Actors:** Customer  
**Description:** A Customer arrives and inserts the card. If the card is invalid the system notifies the user and ejects the card. Otherwise, the customer enters the PIN. If the PIN is invalid the system notifies the user and requests the PIN again. When the customer enters incorrect PIN for three continuous tries, the system retains the card.

**Use case:** Withdraw money  
**Actors:** Customer  
**Description:** If a new customer arrives then he logs in the ATM. The customer enters the amount and the system checks if the amount can be withdrawn. If the amount exceeds the balance, the system notifies the user and requests for a new amount until the condition is satisfied. If the amount is not available, the system notifies the user and requests for a new entry until the condition is satisfied. The system ejects the amount and the card. If the customer does not take card the system acoustically notifies customer and retains the card after a timeout of 30 seconds. Otherwise, the customer takes the amount and the card and leaves.

The above descriptions contain conditional statements describing the alternative behaviors of the system depending on the actor behavior. That is, the main and the alternative scenarios are mixed up. Nevertheless, this is a common description one would expect to retrieve from a user of the ATM.

In the following, the use cases “Withdraw money” and “Login” are written as contents of a use case set in an XML document (Figure 4). This document conforms to the DTD description of Figure 2.

The resulting XML document is relatively legible. The experienced developer should have no difficulty in reading through the file to determine the requirements of the system under development. Nevertheless, it is not acceptable for documentation purposes or for demonstration to the technically inexperienced customer. For these purposes, texts in natural language are more appropriate. Therefore, we present some XSLT transformations that produce different presentations of the same XML file.

Three different presentation formats are presented:

**Simple text format** This is a one column transformation that resembles the structure of the original text format; it merges the main and the alternative scenarios to construct the complete story of the use case (Figure 5). One can compare this text to the original text in natural language, to find out that they are not much different.

**One column transformation** In this transformation the flow of events of the main scenario is enumerated and written. Then, the extensions, also enumerated, follow (Figure 6).

**Two column format in two tables** In this presentation the events initiated by the actor are separated from the system responses, by writing them in two different columns. The flow of events is enumerated. Also, the main scenario and the extensions are presented in two different tables (Figure 7).

```

<?xml version="1.0"?> <!DOCTYPE UseCaseSet SYSTEM UseCaseSet.dtd">
<UseCaseSet>
  <actor name="Customer"/><actor name="System"/>
  <usecase name="Login">
    <main>
      <step number="1" who="Customer ">inserts card</step>
      <step number="2" who="System ">validates card</step>
      <step number="3" who="Customer ">enters PIN</step>
      <step number="4" who="System ">validates PIN</step>
    </main>
    <extension extending="2" number="a">
      <condition>Invalid card</condition>
      <step number="1" who="System ">notifies user, ejects card,
        end of use case</step>
    </extension>
    <extension extending="4" number="a">
      <condition>Invalid PIN</condition>
      <step number="1" who="System ">notifies user, requests PIN again</step>
      <step number="2" who="Customer ">re-enters PIN</step>
    </extension>
    ...
  </usecase>
  <usecase name="Withdraw_money">
    <main>
      <step number="1" who="Customer " includes="Login">arrives and </step>
      <step number="2" who="Customer ">enters amount</step>
      <step number="3" who="System ">validates that the amount
        can be withdrawn</step>
      <step number="4" who="System ">ejects amount</step>
      <step number="5" who="System ">ejects card</step>
      <step number="6" who="Customer ">takes amount and card</step>
    </main>
    <extension extending="3" number="a">
      <condition>Amount exceeds balance</condition>
      <step number="1" who="System">notifies user, requests amount again</step>
      <step number="2" who="Customer ">re-enters amount</step>
    </extension>
    ...
  </usecase>
</UseCaseSet>

```

**Fig. 4.** The use case set consisting of use cases “Login” and “Withdraw money” in XML. (Some extensions are omitted to fit in a page)

## 5 Conclusion

In this paper, we have presented a structure of use cases and modelled this structure in XML by defining the appropriate tags. Our approach provides a number of useful features for modeling use cases.

**Use Case:** Withdraw money  
**Actors:** Customer  
 Customer arrives and Login. Customer enters amount. System validates that the amount can be withdrawn. If Amount exceeds balance: System notifies user, requests amount again. Customer re-enters amount. If Amount is not available: System notifies user, requests amount again. Customer re-enters amount. System ejects amount. System ejects card. Customer takes amount and card. If Customer does not take card: System acoustically notifies customer. System retains card after a timeout of 30 seconds. If 30 seconds timeout: System retains card.

**Fig. 5.** Transformation of the use case “Withdraw money” into simple text format.

**Use Case:** Withdraw money  
**Actors:** Customer  
**Main success scenario**  
 1. Customer arrives and Login  
 2. Customer enters amount  
 3. System validates that the amount can be withdrawn  
 4. System ejects amount  
 5. System ejects card  
 6. Customer takes amount and card  
**Extensions**  
 3a. Amount exceeds balance :  
   3a1. System notifies user, requests amount again  
   3a2. Customer re-enters amount  
 3b. Amount is not available :  
   3b1. System notifies user, requests amount again  
   3b2. Customer re-enters amount  
 6a. Customer does not take card :  
   6a1. System acoustically notifies customer  
   6a2. System retains card after a timeout of 30 seconds  
 6b. 30 seconds timeout :  
   6b1. System retains card

**Fig. 6.** The use case “Withdraw money” in a one column format.

The structured character of XML obliges and at the same time assists the developer to write down all the required fields. It acts as a template with fill-in slots that have to be filled. The resulting XML documents are certainly not formal specifications of use cases, but they follow strict syntactical rules and are checked for consistency. We also presented transformations that allow different visualizations of the same use cases. We discussed further transformations that would allow XML use case documents to be the central repository for a collection of CASE tools, based on XSLT, for requirements engineering.

The presented XML logical structure can be extended to include other kinds of information, which usually do not appear in a use case text, but are associated to

<b>Use Case: Login</b>	
<b>Actors: Customer</b>	
<b>Main success scenario</b>	
<b>Customer</b>	<b>System</b>
1. Customer inserts card	
	2. System validates card
3. Customer enters PIN	
	4. System validates PIN
<b>Extensions</b>	
<b>Customer</b>	<b>System</b>
	2a. Invalid card :
	2a1. System notifies user, ejects card, end of use case
	4a. Invalid PIN :
	4a1. System notifies user, requests PIN again
4a2. Customer re-enters PIN	
	4b. Invalid PIN entered 3 times :
	4b1. System retains card, end of use case

**Fig. 7.** The use case “Login” in two column format.

the use case. An example is the information contained in a system sequence diagram. The use case description can be augmented with this information without destroying the rest of the document. This extra information may be used for the generation of diagrams, design documentation, or even test case sequences.

The production of more transformations and the extension of the logical structure are goals of further research.

## References

1. Anton A., Potts C., *A Representational Framework for Scenarios of System Use, Requirements Engineering*, Springer, London (1998)
2. Biddle R., Noble J., Tempero E., Supporting Reusable Use Cases, In: Springer LNCS Vol.2319, (2002) 210-226
3. Cockburn A., *Writing Effective Use Cases*, Addison-Wesley, (2000)
4. Florescu, D., Kossmann, D., Storing and Querying XML Data using an RDMBS, *IEEE Data Engineering Bulletin*, 22(3):27-34, (1999)
5. Jacobson, I., Object Oriented Development in an Industrial Environment, *Proceedings Conference on Object-Oriented Programming Systems, Languages and Applications (OOS-PLA)*, Orlando, FL, (1987) 183-191
6. Jacobson, I., Christerson M., Jonsson P., Overgaard G., *Object Oriented Software Engineering: a Use Case Driven Approach*, Addison-Wesley, Wokingham, UK, (1992)
7. Rumbaugh J., Jacobson I., Booch, G., *The Unified Modeling Language Reference Manual*, Addison-Wesley, (1999)
8. Schmidt, A. et al, Why and How to Benchmark XML Databases, *ACM SIGMOD Record*, 30(3), (2001)
9. Tuok R., Logrippor L., Formal Specification and Use Case Generation for a Mobile Telephony System, *Computer Networks and ISDN Systems*, 30:1045-1063 (1998)

10. W3C, Extensible Markup Language (XML) Vers.1.0 (Second Edition), W3C Recommendation, (2000)
11. W3C, XSL Transformations (XSLT) Vers.1.0, W3C Recommendation, (1999)
12. W3C, Extensible Stylesheet Language (XSL) Vers.1.0, W3C Recommendation, (2001)