An Extended UML Metamodel for Efficient Application Design and Development

Gabriele Cestra¹², Gianluca Liguori² and Eliseo Clementini¹

¹University of L’Aquila, Department of Industrial and Information Engineering and Economics, Via G. Gronchi 18, I-67100 L’Aquila, Italy ²Bludeep S.r.l.
Corso Umberto I 103, 65015 Montesilvano (PE,) Italy
gabriele.cestra@gmail.com, gliguori@gmail.com, eliseo.clementini@univaq.it

Abstract

Information Systems are becoming more complex as time passes and this complexity needs to be managed. Moreover, the complete matching between applications requisites and delivered software systems is one of the most important elements in software development, since cost effectiveness, efficiency and customer satisfaction are the key of success. In order to achieve these goals, many designers and analysts are applying Model-driven Engineering and UML modelling to their projects, which is gaining an important relevance in the Information Technology environment. This article proposes an extension to the standard UML metamodel with the scope to improve the development process and to grant a high quality of the software delivered to the customers. The metamodel extension contains a collection of elements, with a proper semantic and a set of rules to link them in order to create robust and expressive models, which can be transformed almost seamlessly into code with the help of a Java framework developed in conjunction with the custom UML profile.

Keywords: Model-Driven Engineering; UML metamodel; UML profile

1. Introduction

The increasing complexity of software systems, along with the need to optimize the development process, in the last few years lead to the origin of many analysis and design methodologies, most of them based upon the object-oriented paradigm. One of the most interesting innovation elements in this field is represented by the Model-Driven Engineering [1]. Since its introduction, a big effort has been spent in the formalization of software modelling methodologies and notations. This happened mostly to achieve two important goals: to represent in a rigorous and formal way information systems in order to manage their increasing complexity, but also to automate, as much as possible, the development and code generation process starting from the defined graphical models.

The model-driven software development process represents an approach in which software systems are defined by models and built, at least some parts of them, automatically from these models. A software system can be modeled at different abstraction levels and the syntax for each level is defined by a meta-model.

According to Figure 1 the model represents a software system seen from a certain abstraction level, from which we can generate one or more implementations; the meta-model defines the syntax of a class of models. Every meta-model is related to a particular platform,
that is the execution environment of a software system, for example the Java platform. The metamodel semantic is defined through a set of transformation rules that map each artifact of the language with artifacts of the reference platform: a possible example for this statement is the transformation of an UML class into a Java class [2].

![Model and Metamodel Structure](image)

**Figure 1. Model and Metamodel Structure**

The UML language is the de-facto standard language for the object-oriented modelling [3]. Moreover a lot of modelling tools support UML and a lot of developers are familiar with this language. These reasons lead us to propose a software analysis and design approach based upon the UML language, in order to achieve an almost effortless application of the methodology in IT development processes.

In this article we describe an extension of the UML meta-model that uses the extensibility features provided by UML through a set of predefined constructs called UML Profiles [4]. This extension has been designed in parallel with a development framework [5] based on the Java environment that allows a strong integration between the elements defined during the analysis and design phase and those built during the development phase. The set of formal and semantic rules introduced in the proposed UML Profile represents the meta-model. Then we add a graphical notation based upon stereotypes and attributes. The meta-model is then linked to the framework developed, in Java technology, which can be identified as the platform element.

This extension allows us to apply a modelling methodology that proved to be valuable in the development of different kinds of software systems [6], allowing the traceability of every element belonging to the information system from analysis phase, through the software factory process, to the code writing phase.

In Section 2, we will present the motivations that lead us to the proposal of the extended metamodel in relation to the existing literature. In Section 3, we will describe the artifacts belonging to the metamodel and the rules that they have to obey in order to obtain formally and semantically valid models. Section 4 illustrates the interactions that take place between Use Cases and the other model elements. In Section 5, we will show a model obtained applying the meta-model rules to a real application context, underlining the correspondence between the model and the implementation code obtained using the methodological framework. Section 6 draws short conclusions.
2. Why another UML Profile Extension?

As already stated, an important effort is being currently invested in the model-driven engineering field. The extension of standard UML metamodel with semantically and grammatically enriched profiles is considered a valuable way to define a toolbox useful for the efficient analysis and design of complex software systems. Some authors propose metamodel extensions suited for highly focused applications, such as security [2], data warehousing [7] or System-On-A-Chip (SoC) design [8]. Other authors take a more general approach, extending the metamodel to help applying programming paradigms such as Aspect Oriented Paradigm (AoP) [9] or to develop real-time embedded software systems [10]. The main goal of our research activity was to define a methodological approach to analysis, design and development of rich internet applications. We think that the development process, in order to be as effective as possible should grant the correspondence of model elements during the whole process, trying to minimize semantic and architectural mismatches between model representations corresponding to each software factory phase. This leads to the definition of the extended UML profile described in this article.

The Use Case (UC) can be considered as the central element of the proposed metamodel. The UC construct, which is already present in the base set of UML, in this context gains a much bigger relevance compared to standard approaches. Usually, the UC is defined as an abstract representation of requisites and features of the information system, used at higher levels of abstraction to model the functionality that the system has to provide to the actors. In our work, instead, the UC is a structural part of the software system at every abstraction level: it is defined at the beginning of the analysis phase and it continues to be present during the whole process, arriving to have an exact correspondence in the code thanks to the framework. Around the UC live all the elements that compose the extended UML Profile we propose, which also have a direct and seamless correspondence with coded elements. Another important reason that pushed us to propose this approach is the belief that an effective object-oriented model should map closely the real interaction process between actors, system and business entities: in order to achieve this objective we introduced, among the others, the Worker.

2.1. The Concept of Worker in Literature

The model element worker is necessary to model real phenomena. Starting from the hypothesis that the software system has to match the “real world” as much as possible, the introduction of active objects becomes mandatory. We can trace a correlation between the Worker introduced here and the Service proposed by Ivar Jacobson in his article Object Oriented Development in an Industrial Environment [11]. In that article, Jacobson affirms that it is not enough to use entities and use cases to build a software model, but we need to introduce also a third element called service. Nevertheless in this case the author is motivated by the need of modularization of the use cases instead of representing the real phenomena. The introduction of workers in the architecture proposed in this article, driven by their existence in real world let us indirectly match the goal proposed by Jacobson: system modularization. As a result of the worker introduction we can describe use cases as a set of actions where every action “uses” one or more workers to fulfill its scope. The workers introduce decoupling between system objectives and its implementation. The parallelism between workers and Jacobson’s services gets stronger when the author affirms that “Services are consequently a primary requirement when structuring the system and they are ordering units when offering the system. These two functions of services motivate that Service Modelling is a System Analysis activity and not a System Design activity”. According to the
author the definition of services takes place during the analysis phase and not during the design phase; we can state that the services, or workers in our case, are part of the system independently from the chosen technological architecture or implementation.

In the next section it will become clear how the workers contribute to the realization of use cases and again we can find a strong relation between the concept of worker and Jacobson’s thoughts: “A use case employs a specific set of services. A service can participate in many use cases”. This vision of use case appears to be forgotten in time and we can find clues of it only in the concept of Business Worker proposed by RUP methodology [12]. Nevertheless the RUP presents the Business Worker as an element of the business model, not belonging to the system view, provoking the disappearance of service as an atomic element of the software system (“A service is an indivisible ordering packet of functions for a specific system” [11]). We can affirm that the introduction of the worker as a modelling artefact and its direct mapping and transformation into code represent one of the most important concepts of the proposed methodology and architecture. Later in the article, we will describe in detail the relation between Worker and use case and the worker impact upon the methodological point of view.

3. The Extended UML Metamodel

As already anticipated, the methodology proposed in this article is “use-case driven”. The use case can be defined as “an objective that an actor should achieve by using a complete functionality of the system” [13, 14]. According to this definition, describing a use-case means to identify a sequence of actions that define it. According to other authors, another definition is “A use case is a sequence of actions that an actor (usually a person, but perhaps an external entity, such as another system) performs within a system to achieve a particular goal” [15]. It is clear that a UC is described by actions that a user can activate on the system. Therefore, we think it is possible to represent relations between a Use Case and actions through a class diagram.

3.1. Use Case and Actions

![Figure 2. Use Case and Actions](image-url)

As seen in Figure 2, since the use case is a class of the UML metamodel, it is possible to decorate it with relations with other classes. The effort taken in the complete description of the use case, marking its actions, will be important to build a model useful for the activation of the MDA process described in this article. If we project the diagram seen in Figure 2 in the Object Domain, we can consider the use case as a “collaboration” [16]. We can affirm that in a system described by use cases, whenever an actor makes a request to the system, the system
instantiates a use case and let the actor access to the corresponding action instance. From an implementation perspective we can represent the actions as “inner classes” of the use case and each of them can be implemented as a “Singleton” [17]. It is clear that modelling the use case and its actions as classes the transformation into code is straightforward, nevertheless it is important to note that the “agile” translation into code of the use case is strongly related to the chosen modelling methodology.

Beside the use case, it is important to describe all the modelling elements linked to it. Starting from a generic user interaction with the information system, we can define a collaboration of different entities. According to Robustness Analysis [15] integrated in the ICONIX process [18], it is possible to describe a system through the use of five elements: actors, use cases, boundaries, entities and controls. In the proposed methodology we find these concepts along with some other elements such as Workers, Views and Query Objects. All of these elements will be related to the use cases and allow us to define a complete model of the information system.

3.2. Use Case and Boundaries

Once we have modeled the actions as classes, we need to understand their role inside the architecture that describes the information system. Identifying as the boundary the interface between the system and the outside, we can imagine that the actor (element which is external to the system) could interact with the system through the Boundaries, as represented in the sequence diagram of Figure 3.

![Sequence diagram of Figure 3. Use Case and Boundaries](image)

This representation is valid for every information system. It is possible to define a scenario as the invocation, made by an actor, of a sequence of action of a use case. The called actions will be a subset (at most perfectly matching) of the actions defined by the use case. Following the OOSE [19] approach, it is possible to model the system using three types of elements (UML stereotypes):

a) The information dimension (<<entity>> stereotype) specifies the information held in the system in both short and long term – the state of the system;

b) The behaviour dimension (<<control>> stereotype) specifies the behaviour the system will adopt – when and how the system changes state;
c) The interface dimension (<<boundary>> stereotype) specifies the details for presenting the system to the outside world.

The use of the “Boundary” stereotype in the diagram is coherent with the definition of “Boundary” typical of the OOSE approach. It is important to note that the use case class can be put in the set of classes defined as <<control>>: it is possible to refine the previous sequence diagram adding the <<control>> stereotype, as seen in Figure 4.

![Figure 4. Use Case as Control](image)

Given the behavioral representation of use cases and boundaries, obtained using the interaction diagram, we can introduce a structural representation made with a class diagram. It is also possible to use the prototyping technique to enrich the information content of the class diagram, as seen in Figure 5.

The “call” relation proposed in figure indicates a structural dependency between a boundary and a use case/action. It is easy to infer the behavior of the system from the structural schema: the click on a button will invoke a specific Action on the use case. Once we have introduced the Boundaries we can continue with the description of the methodology describing the Entities and their relations with use cases.
3.3. Use Case and Business Objects

The execution of a use case must lead to a concrete result; in other words it has to extract or modify some information. The information layer of a system is composed of objects. We will call these objects “Business Objects” and tag them in the UML model as <<entity>>. It is possible to define two categories of Business Objects: one called <<entity>> and another called <<worker>>, both of them belonging to the system class set. While the entities may only be manipulated, the workers can instantiate, modify and eliminate entities. This classification is inherited from the real world, in which there are passive objects and active objects that manipulate them. The introduction of the worker has a strong influence on the proposed architecture, since it will lead to the definition of an Extended Model View Control pattern [20]. Given this, the description of a use case through a sequence diagram will be enriched by two more elements, as seen in the example shown in Figure 6.

The role of the use case is to expose to the boundaries “what the system does”, and not “how to do it”, the worker, on the other side, is the part of the system that contains the logic of “doing things”, in other words it knows how to manage the other business objects. This is compatible with the definition of use case given before. Another good point for the introduction of the worker in the model architecture comes from the observation of the real world: in information systems generally the “active tasks” are carried on by a person and not by an object. Figure 7 shows the structural representation of worker and business object.

![Figure 6. An Example of Sequence Diagram](image)
3.4. Use Case and Views

Before starting with the concept of view, it is convenient to integrate the concepts of Boundary, Action, Use Case and Worker in a more detailed and structured overview compared to what done in Section 3.3.
Looking at the diagram in Figure 8 it is evident that a use case can request a certain type of Boundary and some Boundary elements can “call” the execution of actions. You can also notice from the same diagram the strong link between Boundary and use case. We will say that a Boundary set is always distinctly linked to a use case (throughout instantiate relationship). So we can assume that the instantiate dependency relationship has a strongest meaning compared to the common use. In reality the relationship means that there is a navigable structural relationship between Boundary and Use case and Vice versa. All can be resumed as a navigable association between Boundary and use case. We will say that this associations always explicit in the instantiate relationship. Defined the link between use case and Boundary, we need to investigate the way Business Objects, Worker and Entity are part of the structural flowchart.

In Figure 9, it is possible to notice that Boundaries are linked to action representations and views of Business Objects. We will say that each time the actor interacts with a Boundary he is interacting with an entity throughout the actions. Modifying a Boundary means to modify the representation of an entity. As the diagram shows, there is a strong affinity with the MVC architecture pattern. Considering stereotypes <<Control>>, <<entity>> and <<Boundary>>, a very similar MVC pattern scheme would be obtained. The Worker introduces a pattern extension creating the extended MVC. It is important to notice as the <<Boundary>> represents the <<Entity>> “from a certain point of view”. It is necessary to recall the concept of view introduced before. An object can manifest itself in different ways depending on the context. The view concept represents what of the object is visible and reachable, or in other words, editable. Given the need to express in the model all the elements related to the information system, we can model the views as classes belonging to the model.
The view describes how an object can be seen in a particular scenario, independently from technology used to represent it. One view can express, for instance, that a property is mandatory or that is locked (e.g., not editable): the view shows reachability and observability of each attribute of the entity. The GUI of the software system can therefore be properly configured following the description found on the view. A view can describe an a desktop application form as well as an html page or a paper document. In practice views are independent from GUI and not vice versa. Indeed aspects such as a mandatory or read-only attributes depend not only from the view, but also from the intrinsic constraints of the business entities. Observability and reachability of an object depend on specific characteristics of the object along with the object business rules. So there are some rules that are part of the business object nature and cannot be overridden, and other rules that are view-specific, meaning that they are valid only inside certain scenarios. It must be noticed that a view can only narrow these business constraints and not release them. To express attribute constraints of a Business Object it is possible to use Object Constraint Language (OCL) [21]. Using a formal language to express constraints makes possible to execute an automatic transformation into code.

3.5. Use Case and Query Objects

An additional element of the enriched metamodel is the Query Object. A Query Object is required to fetch or select one or more Business Objects from a generic resource. According to the proposed approach it is necessary to place Query Objects in the model. It is fundamental that Query Objects and associated criteria respect the logic of OO paradigm. Figure 11 shows a possible representation of the Query Object structure.
As notable on Figure 11, beside Query Object and Criterion there is another class: the Query Container. This class represents a possible group of queries than can be executed on a particular kind of Business Object. You can observe as each Query Container is dedicated to fetch and select a particular Business Object: this could be considered a very strict rule, but considering the hypothesis that the objects graph is correctly navigable, that assumption is compliant with OO paradigm. A Query Container object will be characterized by $n$-attributes, $m$-queries and for each query $i$-criteria.

As an example we can consider a Query Container related to Person object. We can define two attributes: birthDateFrom and birthDateTo. The filter should be able to describe a fetch operation, from a generic resource, for people born inside a given period. The select condition, that will be implemented as a linear combination of criterion, will be expressed inside the model using the Object Constraint Language:

```java
Person.allInstances->
select(
    p : Person |
    p.birthDate > birthDateFrom AND
    p.birthDate < birthDateTo
)
```

The adoption of an OCL notation to the metamodel for the Business Objects fetch operations has the advantage to be formal and represents an additional semantic step to MDA. Moreover many UML modelling tools offer OCL support, so it is possible to integrate the OCL rules inside the model derived from the proposed metamodel.

![Figure 12. Use of OCL to Model Criteria](image)

An additional aspect of filter modelling through the Query Container class element is the opportunity to represent them with GUI.

![Figure 13. Boundary and Query Container](image)
As it is possible to represent a Business Object inside a GUI element, as seen in the previous section, it is also possible to do the same thing with Query Container. So also for Query container a transformation matrix is required to customize passage from domain layer to presentation layer: for that reason we need to introduce the Query View: the view concept is applicable both to Business Object and to Query Container. It is possible to affirm that, using the proposed metamodel, an applicative domain can be defined by Business Objects and Query Containers.

Figure 14. View Concept applied to Query Container

Query Container represents the link between the generic concept of resource and the application business model. The Query Container modelling completely rescinds from technological implications and from characteristic of involved resources but at the same time it is a bridge to resources world. To better understand as the Query Container is integrated inside the model you can refer to Figure 15. The main rules of analysis keep being valid and the coherency between structural and behaviour views (all the structural elements have a correspondent in the behavior view and vice versa) is assured, allowing us to affirm that the system is strongly cohesive: each elements of the model owns a peculiar responsibility.

Figure 15. Query Container Integration in the Model
4. Use Cases Interactions

In addition to use case collaborations seen in the previous section, the proposed metamodel provides the elements needed to design interactions between actors and use cases and between different use cases.

4.1. Use Cases and Actors

Relationship between actors and use cases represents interaction between actor and system, with the aim to reach one or multiple goals.

![Figure 16. Actors and Use Cases](image)

When there is an interaction between an actor and a system UC, an action is executed on the system, such as pushing a Confirm button on a web page. We can say that each system-actor interaction is based on a request toward the system to do something. Based on this, we can assume that we can have interaction on a system able to do actions. Action definition is integrated in the use case, so each action can be represented by a specific class of each use case.

![Figure 17. Use Case Actions](image)

To interact in a complete way with the actor each use case should be defined as a group of actions:

$$UC:\{A1, A2, A3, A4, A5\}$$

If we take for example a system actor and use case, this can be represented in classes domain or even in the objects domain.
Given the object diagram, it is possible to say that the actor, the relationship and the use case become an actor request, a link and a use case. The link between the actor and the use case request must be considered as an interaction group. An interaction is nothing more than an action call done by the user throughout the system (for instance pushing a button). A scenario is an ordered sequence of interactions:

\[ S = \{I_{1}, I_{2}, I_{4}, I_{fine}\} \]

Scenarios can be interpreted as a complete execution of use case or rather as an use case request. Having defined the use case as a responsibility of the system to produce a valuable result for the user that activates it, we can say that a scenario can be a use case request if it produces a valuable goal for the end user.

4.2. Relationships between Use Cases

The UML defines two kinds of relationship between use cases: Extension and Inclusion [16]. In the presented use case modelling methodology, there is always a collaboration between objects in order to realize a use case. Some of these object could be shared between different use cases. We can consider the example of an Edit use case including a Search use case upon business units of a company. Including use case expects not only a valuable result from the included use case, but also that the use case acts in accordance to his own status. It is mandatory to say that if a use case is in relationship with another, this relationship can’t affect the other use case anyway. We define use case behavior the actions set that the user can activate with it. Using again the use case of the business units search, we can model it as follows as described in Figure 19. We have three actions: “Select”, ”Cancel”, “Change area”. When the use case is included we will have all these use cases actions available to the including use case. Moreover we can say that relationships cannot modify use case behavior. Including use case can “go forward”, throughout the relationship, Business Object and Query Object can be manipulated or used by the included use case as well. It is very important to identify an activation point of the relationship and a call-back point. It is possible to imagine the relationship, in the objects diagram, as a link between two use cases. When we have an including use case this link is in a particular point.
Figure 19. Relationship between Inclusion and Actions

Figure 20. Including Action

Figure 21. Sequence Diagram for Inclusion Relationship
Including use cases leave application flow control to the included use case at a certain point (Extension or Inclusion point) to gain control again at call-back point. We can model these relationships as actions on the including use case, or rather it is possible to say that a relationship means that the use case can activate another use case: in this way the extended metamodel retains all its formal cohesion, as seen in Figure 20. Dependency relationship can be represented by a ternary structural link. Including action can be characterized with Business Object and Query Object that are passed to the included use case, all the structure can be represented by a cascade flowchart (Figure 21). At that point we have an unique way to transform extension and inclusion relationships in codified elements of a model, considering that to each inclusion or extension relationship corresponds an action.

5. Applying the Extended UML Profile

The extended metamodel described in the previous sections has been adopted and applied in different software development activities, mainly in the context of enterprise automation process and management, but also in other fields, such as geo-referenced social-oriented applications [22]. In this section we will briefly describe the model obtained applying the extended UML Profile and its semantic rules in the analysis and design of an Online Learning Management System.

![Figure 22. Online Learning System UC Diagram](image)

5.1. Use Case Diagram

In this diagram we identify the Actors, the Use Cases and the relations between these elements. As seen in Figure 22, the Course Manager and the External User interact with the system through the activation of its use cases. Moreover the model defines some Include relationships between UCs. We have to bear in mind that each element contained in the model derived from the proposed metamodel will have a direct correspondence with an element at code level: in this case each UC will be traced into a corresponding Java class, which is an extension of a class contained in the previously introduce development framework.
5.2. Domain Model Diagram

In the domain model diagram we can find the Business Objects that participate to the realization of the Use Cases (Figure 23). For sake of simplicity attributes and operation have been hidden from the diagram. Nevertheless we have to remember that the aim of the design process is to obtain a semantically complete software model, so the designer will have to define all the operations and attributes features, such as return type, arguments and multiplicity.

5.3. Boundaries and Actions Diagram

In the diagram of Figure 25, we propose the boundaries of the system and the Actions that compose the Course Design UC. Moreover we show the relationship between the boundaries elements that allow actors interaction with the system, such as buttons and sliders, and the Actions that will be activated by those interactions.

6. Conclusions

The proposed metamodel extension allows us to reduce the ambiguity of expressiveness during information systems modelling, providing a set of elements and rules that guide the designer through the development process: in Figure 24 we show these components with their stereotypes. Since the methodology is being applied to different real world software projects, we noted that the ability to trace artifacts all the time simplifies the continuous validation of models: if the model lacks an element, it cannot satisfy the rules expressed in the UML Profile and the problem can be spotted and corrected early in the development process, avoiding the building of incoherent systems that do not meet the functional and not functional requisite of the information system. Moreover, the use-case centric approach provides an
effective way to define the system behavior and structure according to the user interactions. The GUI modelling and its link with the system architecture offers also a strong guideline for efficient prototyping and stakeholder needs verification: it proved to be useful to define detailed UC Scenarios in conjunction with classical Nouns-Verbs analysis approaches [23]. The next step in the research will be a further extension of the metamodel in order to model in detail the behavioral aspects of the information systems.

Figure 24. Elements of a Software System

Figure 25. Online Learning System Boundaries Design Diagram
References

Authors

Gabriele Cestra received his Master degree in Electronics Engineering at University of L’Aquila in 2006. In 2013, he received a Ph.D. in Electrical and Information Engineering in the same university, presenting a thesis entitled "A methodology and a framework for the development of enterprise Rich Internet Applications". From 2007, he works at Bluedeep S.r.l., formerly as software analyst and designer and currently as team leader. His research interests include software analysis and modeling techniques, semantic model transformation and enrichment and mobile computing.

Gianluca Liguori received his Master degree in Electronics Engineering at University of L’Aquila in 2005. In 2010, he received a Ph.D. in Electrical and Information Engineering in the same university. In 2007 he founded Bluedeep S.r.l., in which he currently holds the position of CEO. His research activity is mainly concerned with information systems and software engineering, especially related to software development methodologies.

Eliseo Clementini is an Associate Professor of Computer Science at the Department of Industrial and Information Engineering and Economics of the University of L’Aquila, Italy. His research activity is mainly concerned with information systems and databases, especially related to spatiotemporal data. He has active scientific collaborations with many international experts in the fields of spatiotemporal databases and qualitative reasoning. He was a visiting researcher at the INSA of Lyon, at the University of Liège, at the University of Glasgow, at the University of Maine, at the Technical University of Munich, at Hiroshima University, and at Simon Fraser University. He has been an invited speaker in various international symposia, such as the SAGEO 2012 conference. He is a member of the program committee of several conferences such as COSIT (Conference on Spatial Information Theory), SDH (Spatial Data Handling), ACM SIGSPATIAL GIS (Advances in Geographic Information Systems), and SeCoGIS (Semantic and Conceptual Issues in GIS). He is in the editorial board of the International Journal of Spatial, Temporal and Multimedia Information Systems.