# SOFTWARE DEVELOPMENT GUIDED BY MODELS
## *The XIS UML Profile*

Miguel Luz, Alberto Rodrigues da Silva
*INESC-ID, Instituto Superior Técnico, Rua Alves Redol 9 , Lisbon, Portugal*
*Email: miguelluz@netcabo.pt, alberto.silva@acm.org*

Keywords: UML, XML/XMI, UML profile, Computer software models, Generative code

Abstract: The UML is used to detail high level software specifications that will be interpolated for XMI and XIS (XML Information Systems) as interchange formats based on XML. UML and XML are expected to be the next generation of modeling and data interchange standards respectively. In this paper, we describe the UML Profile for XIS architecture as a proposal for software development guided by UML models. The XIS system is based on a multi-phase generative programming approach, starting from high-level UML models till software artifacts (such as Java code and SQL scripts), passing through different representations, namely OMG's XMI, and our (XIS) specific XML vocabulary. The main contribute of this paper is the overview of the XIS system and the proposal and discussion of the XIS UML profile.

# 1. INTRODUCTION

In the last years, the software engineering community has been coming to try to establish a standard for the software development process. MDA (Model driven architecture) [15] is perhaps the most likely attempt to reach that objective, of establishing a true standard capable to give scalability, compatibility and portability warranties to who uses it.

This work appears in this context, as a vertical proposal for the software development based on high level specifications supplied by UML models. The XIS (XML Information Systems) project [25] takes advantage of technologies thoroughly spread and used by the MDA model, such as UML and XML/XMI.

In Section 2 of this work we present and describe UML as a systems modeling standard and equally describe the extension mechanisms that allow us to create a UML profile for the XIS architecture. The technologies related with the transfer of information, that is to say, XML and its derived language XMI, will be presented in a summarized way in the Section 3.

In section 4 we present the above referred MDA and equally UIML (User interface markup language), the last one as a user interface representation model, totally independent of platforms or devices.

The XIS architecture and its UML profile [17] is proposed and described in the section 5 and is based on a deepened study of the competitive profiles already presented as the UML profile for Web applications [1] or the UML profile for XML Schema [3],[7],[10].

# 2. UML, MOF AND EXTENSIONS MECHANISMS

## 2.1 UML

UML (Unified Modeling Language) [2],[4] derives mainly from the unification of three methods: "OMT" of Rumbauch, the "Booch method" and the "use cases" of Jacobson. Each one was well succeeded as an object oriented analysis method therefore UML takes advantage of the main characteristics of each one. UML is a model representation language (object models) that doesn't define a standard process but is intended to be useful with an iterative development process.

As all the languages, the UML is used to transmit to others, and to receive from them, some information. In that case the information is the precise definition of a system and how it should be implemented to accomplish what the user wants.

Each concept to be modeled (subsystem, class, relationships, etc) has a specific graphic representation in the language. It is very important to respect the defined UML conventions so that other people can understand the diagrams. The UML can be used with any process of software development. The UML notation is big and flexible enough to supply the needs of an enormous range of projects. It's very important to choose which specific element type of UML notation will be necessary depending on the nature of each project (client/server application vs. a real time application). The choice of which implementation language should be used (notice that UML is language independent, but which structures will be used, those yes, depend on the language)

it's equally important. For example some details of C++ construction won't be necessary if the used language is Java or Smalltalk. In spite of that, it is important to notice that it is possible to model basically any "Entity" using UML.

## 2.2 MOF

The MOF (Meta Object Facility) [13] defines a simple meta-metamodel (also called MOF model) with enough semantics to describe metamodels in several domains, being the initial focus analysis metamodels for OO projects. The integration between metamodels is done by the interfaces also offered by MOF, being necessary for the integration of tools and applications (for the several phases of the development life cycle) using a common semantics.

The main objective of MOF is to offer a framework that supports any metadata type and that any metadata type can be added when necessary. To do that MOF uses a metadata multi-layers architecture based on the traditional metamodeling four layers architecture (very popular inside the standardization communities as ISO and CDIF).

The key feature of that architecture is the meta-metamodeling layer that supplies a common language that joins metamodels and models.

The highest layer - M3, is composed by the MOF model that is for example instantiated in metamodels for UML or OMG [21] IDL in the level M2 and these are for its time instantiated in the UML and OMG IDL models respectively in the layer M1.

The MOF model is object oriented and its metamodel constructs were defined using UML (and therefore the UML descriptors).

The MOF model is self descriptive that is to say the MOF model is formally defined using its own constructs. That characteristic of being self descriptive facilitates the definition of MOF interfaces and behaviors by applying the MOF IDL mapping to the MOF model, offering this way semantic uniformity among computational objects that represent models and metamodels. This also means that when a new mapping technology is defined, APIs for mapping metamodels in that context are also defined implicitly.

UML as we have seen is a modeling language object oriented (UML is not a methodology) defined by MOF. It is not a proprietary notation, being this way accessible for everybody – toolmakers, training centers and others that can use it gratuitously. UML is generic, rich, simple but not semantically simplistic. Five groups of diagram are offer by UML, they are: use cases diagrams, class diagrams, interaction diagrams (sequence diagrams, collaboration diagrams), state diagrams (state diagrams activity diagrams), architecture diagrams (components diagrams, installation diagrams).

Between all these diagrams, what interests us in the context of this work and thinking on MOF, are fundamentally the class diagrams. The class diagram exhibits the classes that will be included in its application and the relationship among the classes. Several class diagrams exists - high level class diagram that identifies some classes and the basic relationships among them and later an entire series of more complex class diagrams that incorporate more and more information. When a modeling becomes very big, it is probable that can be divided in modules. In UML the modules call themselves packages, exists a class diagram for each package. To define metamodels, MOF uses a group of metamodels constructs that have the 4 main concepts of OO modeling: classes, associations, data types and packages (Package):

?? Classes: model the MOF metaobjects;

?? Associations: model the relationships (always binary) between metaobjects;

?? Data types: can have two purposes: define types whose values don't possess object identification (e.g.: integers, strings,...) and reuse of external types defined in some interface specification not MOF (in the present version it only supports CORBA data types)

?? Package: is the construct of the MOF model that contains elements in a metamodel. In the layer M2, the Packages split themselves and they allow the construction of modules in the metamodel. The packages can contain other packages, classes, associations, data types etc.

The communication between the layers of the MOF architecture is crucial for the mapping among layers.

## 2.3 UML Extension Mechanisms

Evolutionary potential, not only the adaptation capacity, but also the capacity to develop starting from adaptation, is the primordial weapon to prosper in a changing environment with growing complexity. The language used to communicate and interact is extremely important, because it provides the mean for communication and collaboration among members of a same group. The languages establish the ground base on which groups are erected and on which are developed. The language itself is not only the mean of communication but also the structure for individuals' experiences and knowledge representation inside a group.

UML is a generic tool with evolutionary capacity, supported on its own tools. The language is thoroughly applicable to different types of systems (software and other), domains (business, industry), methods and processes.

UML provides the capture, communication and knowledge distribution: the models capture knowledge (semantics), the architecture vision organizes the knowledge in agreement with directives that express its use, and the diagrams represent the knowledge (syntax) for communication.

A language consists of a collection of concepts (semantics) with a notation (syntax) and rules that guide the concepts and the notation. Underlying a language and the methods that use that same language is a base constituted by fundamental beginnings or axioms. These fundamental beginnings involve elements

"universally" accepted as essential or elements that define the construction on each language have sense.

The UML extension mechanisms are the essence that facilitates the language to adapt to different types of systems, domains, methods and processes. These mechanisms should be necessary, enough, and consistent to establish a robust way to extend the language. The process through which is evaluated the necessity, sufficiency, and consistency of the UML extension mechanisms UML involves:

?? The knowledge and understanding of the conceptual structure, generally used in modeling.

?? The knowledge of the UML extension mechanisms as well as the rules that regulate the use of the same mechanisms.

?? The knowledge of how the application of the conceptual model is accomplished for modeling; and to apply UML to determine some of the rules that regulate the use of the UML extension mechanisms is problematic, contradictory, or inconsistent.

?? Summarizing, detection of any inconsistencies inside the rules that govern the use of the UML extension mechanisms, as well as to evaluate its indispensability.

To guarantee the liability of this evaluation, multiple examples and sceneries are foreseen and approached, including in this analysis, very detailed diagrams and minimal detailed diagrams. Very detailed diagrams portray the whole notation on the other hand minimal detailed diagrams describe only the necessary notation.

The extension mechanisms are the mean through which we can adapt and extend UML action scope. UML defines properties for each element of the model and means to add new types of elements and to modify properties of existent elements.

Stereotypes are used to classify or to mark model elements and to introduce new types of model elements. Each stereotype defines a group of properties that are tied up to the elements of that stereotype; on the other hand it defines "well formed ness" rules that will have to be satisfied for the elements of that stereotype. Stereotypes are used equally to create metamodel elements. The stereotypes are represented by a chain of characters, begun with "«" and finished with "»", that precedes the name of an element. The stereotypes can also have an associated graphic icon. Other possible extensions are the Tagged Values and Constraints.

**Evolutions of the pattern for UML 2.0**

UML is a language in permanent evolution in order to handle the demands of the software development community. OMG centralizes the community requests and creates work groups to study and to proceed with the necessary alterations, for then later to publish the results in a new one standard. This process is highly controlled by to guarantee that great distortions are not generated, from a version to another, because that would imply changes possibly not supported by all the links of a chain of software development (e.g.: development tools, methodologies, etc). This practice taken place by OMG allows the diffusion and effective use of a standard, even in long term projects, once it protects the know-how "investment" of the user, this way he can concentrate on the study of its project and not in the permanent upgrade in the modeling language.

For the elaboration of version 2.0 of UML [23,24], there were constituted, in the middle of 2001, four working groups, that should finish the new purpose during 2002. Some of the objectives they intend are:

?? Improving the extension mechanisms;

?? Possibility to define new diagrams in the UML Profiles;

?? Pluggable Systems components modeling and patterns for systematized reuse;

?? Creation of a new semantics for state machines;

?? Improving the events management and the management of data flow in the activity diagrams;

?? Notation for patterns

In parallel with the specification UML 2.0, is in phase of evaluation numerous profiles among which stand out: Profile for Data Modeling [6], Profile for Web Modeling [3], Profile for Real-Time Modeling [18] and the Profile for Agent Modeling [19], besides constant improvements in the profiles already approached in this paper.

# 3. XML AND XMI

## 3.1 XML

XML (Extensible Markup Language) [11],[12],[22] is the new standard, adopted by World Wide Web Consortium (W3C)[20] that complements HTML in data exchange in Web. Both languages are based on the SGML language (General Structured Markup Language). The distinctive characteristic of SGML is that the format and the structure of the document are defined through elements denominated tags. We can identify the tags in SGML by the symbols '<' and '>', as for example in the tag <SGML>.

The main difference in relation to HTML is that XML doesn't describe the presentation of a document, but its content. Another important difference is that XML is expandable; it allows the creation of new tags while the group of HTML tags is fixed. XML tags can be chained in any depth level; HTML allows, but it doesn't recognize, the linkage of tags. The last, but not less important characteristic of XML is that it allows to be defined a grammar (DTD) for document validation while HTML by not allowing the creation of tags, doesn't allow another grammars different from its own.

To well understand XML is important to consider its ambivalence as language for document structuring and as language for content definition. XML, as well as SGML, was thought originally as a language to define structures of documents, independently of its presentation. However, the use of XML to define contents, the "pure" XML or variations like RDF, is quite recent. Thus, there is some characteristic of XML that are important in the documents structuring but are not appropriate for contents definition. An example of that is the imposition of order of the elements that is fundamental to structure documents, but too restrictive to represent contents. Many languages came from XML core, an example of that is the XMI (XML metadata interchange) which is extremely important in the software development subject as we will see further in this paper.

## 3.2 XMI

A problem exists with the present object oriented development tools: the metadata exchange between tools is still a very difficult task. Not only tools store and codify its metadata in different ways, but also the conceptual modeling schemas offered by those tools is also quite different. Then what we have is an interoperability problem in two levels: the level of codifying and the level of conceptual schemas.

XMI (XML Metadata Interchange) [14] is a standard for exchange of object oriented systems models that tries to solve the interoperability problem in those two levels through the definition of a generic code pattern, XML [8],[9], and with the definition of a pattern for conceptual schemas, called MOF.

Standard XMI was created with the main goal of allowing the interoperability among CASE tools, metadata repositories and development tools, through the metadata exchange in a file or flow (stream) of data based on standard XML.

In the conceptual level, XMI is based on other pattern called OMG MOF. MOF as we have just seen is a pattern for defining program interfaces for model repositories, but is also a pattern to describe metamodels. MOF is generic and rich enough to be capable to describe appropriately any object oriented metamodel as for example UML.

At the codifying level, XMI is based on XML. But XMI is not a XML language. XMI is a specification of XML to generate XML languages adapted for data models as well to code these metadata in a XML document. Then, the XMI specification is nothing else that a group of rules that normalize the generation of XML starting from MOF.

XMI specification is composed by two groups of rules: a group of rules of XML DTDs production and a group of rules of XML documents production. The first group describes how to derive the XML language grammar corresponding to the metamodel DTDs. Therefore, they describe the rules for the construction of the XML document corresponding to the model. But, due to the little express ability of the DTD's language, it is necessary an additional level of rules to guide the generation of the XML documents. That is the function of the second group of rules.

The XMI rules for automatic DTDs generation create conditions for the production of new types of documents with the purpose of data and semantics exchange.

As an example, a XMI DTD generated for UML allows the exchange of UML models and also of the class definitions. Therefore, a UML model is capable of being changed among project tools and IDEs using UML DTD. The XMI architecture DTD supplies the necessary infrastructure for an advanced information transfer for the uniform treatment of the object identity, internal and external references, document partitions, specific tools extensions, incomplete models and differences. However, only the standardization of DTDs is not enough for the exchange for the most common cases. Observe that DTDs doesn't possess the capability of expressing the semantics adapted for a model. A group of additional concepts is necessary, being available through a complete and wide scope architecture including UML, MOF and other patterns developed by OMG. The UML models and similar should be sources for patterns. Once information that needs to be changed is expressed in UML, XMI automatically can produce DTDs and transfer formats.

# 4. COMPUTER SOFTWARE MODELS

## 4.1 MDA

MDA (Model Driven Architecture) is a new paradigm in the systems specification [15]. MDA can specify systems at all levels, including middleware levels.

This new model of systems architecture, offers us a group of very important advantages for distributed environments:

?? Supports the whole development life cycle with more precision;

?? Decreases development costs;

?? Applicable to all languages, platforms, operating systems, networks and middlewares.

?? Most of the MDA patterns is already available: UML, MOF, XMI, CWM;

?? A powerful middleware infrastructure: CORBA, IDL and services; built on a success OMG platform technologies OMG, like CORBA and UML;

?? Rigorous mapping in the future for another infrastructures as: XML, SOAP, Java, .NET;

MDA being independent of middleware, languages or systems (language -, vendor - and middleware-neutral) guarantees that following its requirements and good development practices will have an application with scalability and migration potential.

In the Figure 1 we can see the several implementing layers or levels of the architecture.

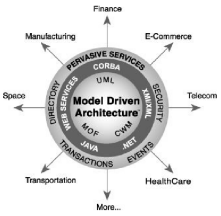


Figure 1: MDA implementing layers

The architecture core that we can see in the center of the figure is based on the OMG systems modeling standards such as UML, MOF and CWM . Several models are foreseen for the core: one that represents enterprise component with structure and transactional interaction; another for real-time systems with specific needs for resources control; more models will be introduced for filling the needs of specific domains. OMG wants to maintain a small number of core models, once these by itself represent in its category all the characteristics of all the possible platforms.

If we have as final objective a CCM, EJB or MTS architecture the first step when creating an application based on MDA, will be the creation of a UML model, derived from the appropriate core model that is independent of the platform. Specialists of each platform will adapt that model (platform independent) to the wanted platform. In the future will appear standards at the level of the automatic mapping of these models in applications you specify for a given platform. This transformation can be understood as the passage of the core of the illustration for the ring next to it. Although the objective is to maximize the automated mapping step, in most of the cases it will be necessary the intervention of the human hand, at least while the tools based on MDA don't reach its grown-up age.

After the mapping stage we are now in the presence of platform adapted models that faithfully represents the semantics of the business model and the technical characteristics of the future application. The next step will be the code implementation in the wanted language.

MDA represents nowadays the next generation of Internet ORB's integrating whole platforms and middleware's, from past, present and future. The group of advantages of the use of MDA as standard is:

The programmer will be capable to develop applications based on MDA, using for such the middleware environment that its company adopted. He will have the warranty that the essential semantics of its application will be translated systematically in a platform independent model and that future migrations for new middleware versions will be relatively easy. On the other hand, any communication attempts inside of the same company or with vendors, customers and partners can be developed falling back upon a consistent architecture and with a certain degree of automatic generation.

?? The key of the business logic applications after having adapted the MDA model can stay in the original platform, MDA will help to create automatic bridges between the several platforms.

?? The futures industry standards will incorporate defined MDA models that will be in fact platform independent.

?? As new middleware platforms emerge, the process of OMG standardization will go on and incorporate them in MDA, defining for each one a new mapping. The tools based on MDA will be this way capable to enlarge its scope of platforms for which can convert PIM's (Platform Independent Model).

?? The programmers will win flexibility and capacity of code restructuring, being based for such in PIM, as the infrastructure of the inferior layers is going suffering alterations.

?? The models are built, visualized and manipulated through UML; Transmitted through XMI; and stored in MOF repositories.

?? The formal declaration of the systems semantics (during the modeling process) will increase the quality of the software as well as increase the systems lifetime.

## 4.2 UIML

A growing number of services have been coming to be offer in our computers. Portable devices and telephone are nowadays more and more used as means of information and communication between users and central servers. This type of devices surpasses billion units nowadays, not limited to the data transfer (climate, news, etc); they offer services today with very more interactivity as home banking, trip reservation and e-mail.

Nowadays we have four categories of devices capable of running a user interface generated by computer: desktop computers, palmtop's and pocket pc, smart phones and traditional phones. Desktop computers are all those devices with graphic displays, a lot of memory and computing capacity, voice capacities and the possibility to carry, to keep and to run executable code (e.g.: Java - applet, Active - X). In contrast, the palmtop's and pocket computers have small displays and limited computing power, but they usually allow carrying small segments of executable code (e.g.: the applications are divided in small segments that are loaded "on demand"). In the case of the smart phones, we are looking for an user interface with a display with some text lines and not much or any computing power. Finally, the traditional phones use recorded sounds and voice synthesis as output and voice recognition for input.

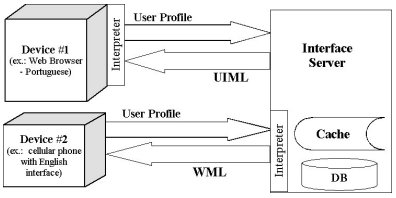


Figure 2: UIML deployment diagram

#### 4.2.1 Problems of user interface development

The user interfaces are implemented through a growing number of different software and hardware technologies. The user interface programmers have for this fact to work with a great number of problems. For example, they have to know and to learn multiple languages: Wireless Markup Language (WML), C++ with specific interfaces for certain operating systems (e.g.: Windows CE or UNIX) among another. To complicate more the process, these languages are recent and are in constant evolution, what forces the programmers to constant upgrades. Parallel to this problem the one of coexistence of code bodies of a language appears inside another. All these problems harm the final objective of consistency and portability among devices.

There are may languages to describe user interfaces namely:

?? Markup Languages for Graphical User Interfaces (GUIs) - PCs with high resolution monitors and with CPU and enough memory can support many user interaction types. GUIs are very rich interfaces that embrace elements as windows, icons and menus for interacting with the user. Markup Languages for GUIs (e.g.: HTML) describe the presentation and layout of the display and use scripting languages to specify certain behaviors.

?? Markup Languages for Palmtop's or Wireless Devices - Markup languages for palmtop's or wireless devices keep in mind three factors: display size, limited bandwidth and capacity of data input. CompactHTML is a subset of HTML for small devices (PDAs, smart phones, etc.).

?? WML - WML (Wireless Markup Language) specifies contents and user interfaces for portable devices, as palmtops or pocket computers, that have connections to wireless networks, including cellular phones and pagers. WML uses the metaphor "deck of cards" to specify an user interface it specifies. The user navigates like this in a group of cards according to a logical defined and organized in decks being able to interact inside each card through menus or text fields. WML includes support for images, text, formatting (bold, italics, etc.) and layout (line, controls, break). It is also contemplated runtime mechanisms that include navigation between letters, links, event treatment etc.

?? Markup Languages for Voice - in a lot of situations the voice is the most effective way to communicate, for example when we are driving or operating industrial machinery. Two languages dispute this space today, IBM' SpeechML and Motorola's VoxML. SpeechML describes applications in pages, bodies, menus and forms. VoxML describes applications in terms of dialogues and steps. These two languages are to be melted in VoiceXML, that this to be standardized by VoiceXML Forum. VoiceXML is a language for specification of applications with voice input/output. It was drawn to create dialogues with capacity of voice synthesis, speech recognition, and recognition for Dual - Tone Multi - Frequency (DTMF) and voice recording.

UIML is a universal and device independent markup language, isolating like this the designer from the different devices peculiarities. It was drawn to establish a natural barrier between user interface code and the remaining, facilitating this way the reuse of the interface and reducing the development time as well as of prototyping. It equally increases the safety and it supplies scalability, indispensable to new technologies adaptation.

A UIML document can be used in multiple ways. It can be kept in a server, and when an user invokes an application, the UIML document is compiled for the platform language (e.g.: for WML or for C++). On the other hand it can be interpreted as the user interacts with the interface, the same happens with the HTML documents when interpreted by a Web Browser. The compilation beside the server is obligatory for devices unable to accomplish downloads, as cellular phones, or for devices with weak resources in terms of memory.

The interpretation is more flexible, for example, the Java interpreter can allow that an interface defined in UIML appears like a Java Bean, allowing the application logic to manipulate it in its interior.

UIML can also be used without a server; in this case UIML or the compiled code is installed together with the remaining of the application in the user terminal. The application logic interacts directly in runtime with the interface. The advantage of running an interpreter on the client's side is of avoiding sending great volumes of executable code for the devices. UIML is comparatively a small text file with the equivalent executable code; the same happens in the case of the HTML forms that consume less bytes than the applet to implement an equivalent form. Other great advantage is the safety, once UIML is a declarative language that is to say it describes what should happen not revealing how it should happen. UIML for itself, like HTML, is less probable of containing a virus or to throw an attack to a client than executable code. However the scripting languages or the connection to the application logic used in UIML are vulnerable to these problems (as in HTML with CGI scripts or the client's scripts).

# 5. XIS SYSTEM

## 5.1 Overview of the XIS System

XIS (XML Information Systems) is the acronym chosen to represent a research project that intends to conceive, to develop and to evaluate mechanisms of information systems production, in wide scale, based in models and software architectures. XIS project intends to study mechanisms of fast conception, modeling, production and test of information systems as well as artifacts generation, reverse engineering and round-trip engineering. The language chosen for modeling is UML. It is this way also a requirement the definition and integration of UML extensions to standardize the generation of systems using XIS architecture.

The XIS system involves a multi-phase generative programming approach as suggested in Figure 3: (1) high level UML specification; (2) transformation of UML models to XIS vocabulary; and (3) code generation.

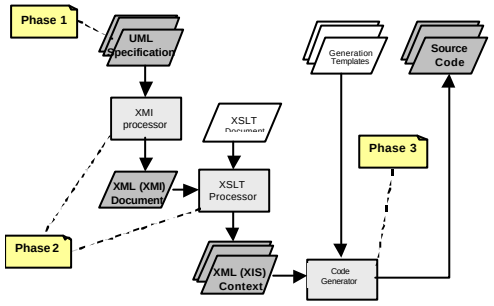


Figure 3: XIS Profile Virtual MetaModels

The first phase concerns the production of IS models using UML and a set of specific UML extensions defined in the UML profile for the XIS architecture (see Section 6). The second phase concerns the transformation of UML models to a manageable format: the XIS XML format. This transformation involves two successive sub-transformations:

?? Apply a XMI processor in order to map UML models into XMI format (a XMI processor is a common component provided by the majority of CASE tools – see Section 3.2);

?? Apply a XSLT processor (developed in the context of the XIS project) to transform the models from the XMI into XML XIS format.

The third phase applies a set of code generative techniques in order to generate code (and eventually other artifacts) from the models represented in XML XIS format.

The representation of UML models in XMI is something extensive and complex, bringing problems in its manipulation. With the project requirements in mind, the XIS vocabulary was created, to easily understand documents which are generated starting from XMI documents, through the specification of the transformation, supplied in XSL.

## 5.2 MVC Architecture

The whole development is based on the architectural MVC (Model-View-Controller) model that will deepen further on this paper. Basically the MVC model separates the system in three different but interlinked cooperative areas, as we can see in the Figure 4.

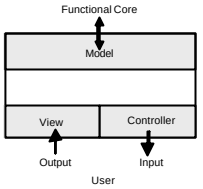


Figure 4: XIS Profile Virtual MetaModels

Graphic user interfaces change frequently, for instance due to modifications in the functionality that requests a change in their behavior; due to needs of alterations in the representation of the data; or due to requirements for new software platforms. We defined in the XIS project the following general rules::

?? Information can be represented in different ways and in different windows.

?? Dynamic feedback: the representations of the information should display, immediately the changes that can happen in the system.

?? Changes in GUI should be easy to handle and some of them in execution time.

The MVC architecture divides an application in three abstractions which are model, view and controller. The model abstraction defines the semantics of the application, it maintains its state and it defines its behavior. The view abstraction allows a form of visual presentation of the model for their application users, the same model could be associated to several views. The controller abstraction defines the glue between models and views, the model component encapsulates the core of the functionality and the data that it manipulates. It is independent of the specific representations presented in the output. View component shows information to the user starting obtaining data from Model. Multiple visualizations of the model can exist. Each View possesses an associated Controller. Each Controller receives events of the entrance and translates it in services for Model or for View.

Model encapsulates the data and exports the functions that offer specific services. Controllers call these functions, depending on the user's commands. It is role of Model to supply functions for the data access that is used by View to be shown.

The changes propagation mechanism maintains a registry of all the components that are dependent of the model components. All the View components and some selected Controllers registry that want to be informed of the changes in Model. Each View defines an update procedure that is activated by the changes propagation mechanism. When this procedure is called, View

component recovers the actual values of the model and it shows them in the display.

Each one of these abstractions (model, view, control) corresponds to an abstract class in the MVC framework, and cooperates through a defined interaction protocol. The classes are defined in the following way:

?? Class Model: a Model object can maintain dependent objects (views and controllers), and send notification messages of modifications in the model for the dependents. The class Model defines a specific messages protocol. A model of an application is an instance of a subclass of Model that should implement specific application behaviour.

?? Class View: a View object can be decomposed in another View objects, called sub visions. The View class defines the protocol for interacting with controller and model objects, interacts with its sub visions, coordinating transformation actions and presentation. A View object in an application is an instance of a subclass of View. The subclass has to supply the implementation for the presentation protocol, to define specific details of the models that it represents.

?? Class Controller: defines the protocol to manipulate Model and View objects, starting from user interaction correspondent messages through data input devices.

The component of code generation, although isn't the focus of the present work, it will also be briefly described in Section 5.3.

## 5.3 Code Generation

The code generation as depicted in Figure 3 is an indispensable task related to the construction of a information systems. Significant advantages exist in the use of code generators namely [5]:

?? Reduction of programming mistakes, leaving free time for programmers to concentrate in the understanding and specification of their systems. Specifications are generally much more accessible to analyse, to write, to publish and to detect mistakes than the code that implements them. The specifications can be analyzed like this and validated by a larger universe of participants.

?? Applications maintenance can be done by people without technical skills.

?? Decrease of the temporal period between prototyping and testing new specifications. When introducing new requirements in the specification, they can be mapped immediately to new source code for testing.

?? Generation of an application family, taking to the creation of variants of the same application, each one with advantages and disadvantages in different environments and situations.

?? Code optimization: if specifications are sufficiently detailed, it will be possible to create optimized code for a certain context.

?? There are also some disadvantages related to the adoption of generator techniques such as:

?? A specific code generator can only be used efficiently, in some situations (e.g.: structural information should be static during the application execution).

?? The creation of code generators requires careful analysis of the specification languages and user interfaces, deep knowledge of the domain of the application and capacity to create generic units of software reliable for different application domain.

?? The recognition that a code generator can be useful in certain situation generally makes late itself, existing little motivation then to re-do the whole system.

# 6. XIS UML PROFILE

This section defines the XIS UML profile, defined in terms of the UML extension mechanisms, namely Stereotypes, Tagged Values, and Constraints. (See the UML Semantics document [16] for a full description of the UML extension mechanisms.) This Section describes stereotypes that can be used to tailor specification of information systems production throw the XIS architecture approach.

All the UML concepts can be used for the referred purpose, but providing specific stereotypes for some common situations provides a common terminology for this domain. Note that UML can be used to model different kinds of systems: software systems, hardware systems, and real-world organizations. This paper is not meant to be a complete definition of XIS profile concepts and how to apply them, but it serves the purpose of overview these proposed extensions, including its icons.

**Profile vs Metamodel**

UML profiles modeling capacities:

?? Structuring the extensions (Packages)

?? Defining new meta-classes (Stereotypes)

?? Defining new meta-attributes (tagged values)

?? Defining new meta-associations (tagged values, referencing to other model elements)

?? Defining new constraints

?? Modeling graphically profiles

This is almost all we need for defining metamodels, but metamodels bring other problems:

?? Troubles with different versions of UML, becoming even harder when combined with MOF/XMI versions

?? Moving from one metamodel to another is a real heavy task, hard for tool implementers, heavy for end users

**Metamodels advantages:**

?? Metamodels are stable (standardized). They do not evolve, or do change only after a long stable period

?? Metamodels are formal : there semantics are completely defined, in a precise and unambiguous way

Weighting profiles and metamodels advantages and disadvantages, our choice redounds on the creation of the XIS UML Profile.

**The reality is:**

?? End users want a stable root standard

?? The extensions that we define are incomplete, informal, and may even be contradictory

?? We need flexibility, ability to change fast, to combine different views

## 6.1 The XIS Virtual Metamodel

A virtual metamodel (VMM) is a formal model with a set of UML extensions, expressed in UML. The VMM for the XIS UML Profile is presented as a set of class diagrams.

The VMM represents a Stereotype as a Class stereotyped «stereotype». The Class that represents the Stereotype is the client of a dependency stereotyped «base Element», whose supplier the UML metamodel element is being extended.

Because Stereotype is a generalizable Element inheritance hierarchies of Stereotype instances can be constructed, and a Stereotype can be designated as abstract. This VMM makes use of these capabilities. The VMM represents a Tagged Value associated with a Stereotype as an Attribute of the Class that represents the Stereotype.

The VMM can also specify Tagged Values that are not related to any Stereotype. The VMM represents this kind of Tagged Value as an Attribute of a nameless Class stereotyped «Tagged Values». The Class stereotyped «Tagged Values» is the client of a dependency stereotyped «base Element», whose supplier the UML metamodel element is being extended. When multiple Tagged Values extend the same base element of the UML metamodel, all of them may be grouped together as Attributes of a single «Tagged Values» stereotyped Class.

The VMM adds new Packages to the base UML 1.4 metamodel. These Packages contain the Stereotypes and Tagged Values that make up the profile. The XIS profile packages are depicted in Figure 4.

The VMM depicted in Figures 5 to 7 are the correspondent to the Model, View and Controller Packages.

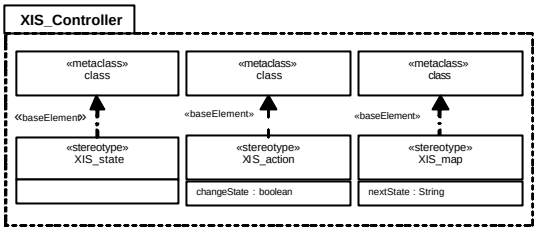


Figure 5: XIS profile Controller Package

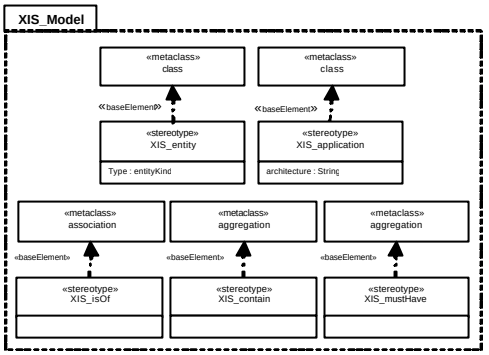


Figure 6: XIS profile Model Package

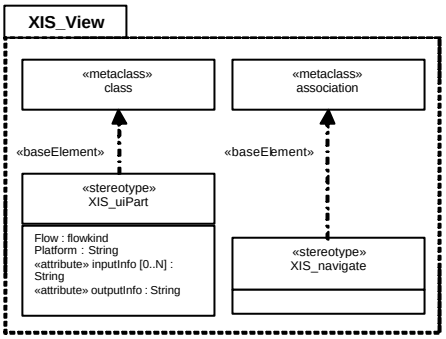


Figure 7: XIS profile View Package

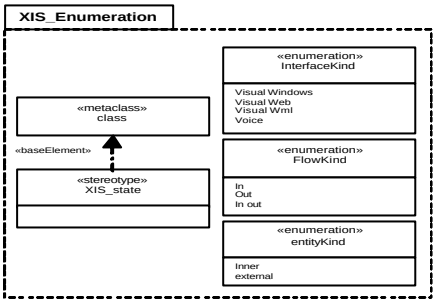


Figure 8: XIS profile Enumeration Package

The UML extension mechanism permits the creation of symbols to represent defined stereotypes. The primary goal for this is to turn a model more legible therefore the defined symbols for the XIS architecture profile are depicted in Table 1.

| | |
|---|---|
| «XIS_entity» | «XIS_state» |
| «XIS_enumeration» | «XIS_action» |
| «XIS_uiPart» | «XIS_map» |

Table 1 – Stereotype Symbols

Apart from standard UML combinations the following (Table 2) combinations or better known as well formed ness rules are allowed for each UML profile for XIS stereotype:

| from: to: | XIS_application | XIS_entity | XIS_uiPart | XIS_state | XIS_action | XIS_map |
|---|---|---|---|---|---|---|
| XIS_application | association | association | - | - | - | - |
| XIS_entity | association | isOf<br>mustHas<br>contain | association | - | - | - |
| XIS_uiPart | - | association | navigate<br>contain | mustHas | mustHas<br>contain | - |
| XIS_state | - | - | - | - | - | mustHas |
| XIS_action | - | - | mustHas | - | - | mustHas |
| XIS_map | - | - | - | - | - | |

Table 2 – Well Formed ness Rules

As we have seen the new XIS architecture for information system development expands the UML standard profile, while introducing some but important changes in the conceptual definitions of the class stereotypes. Therefore we present the modified definitions of the UML profile for XIS:

**Class Stereotypes:**

?? «XIS_entity»: the entity class is used to model persistent information. Entity classes structure the field of action classes, often representing a logical data structure. Entity classes are the backbone of the application, in the figure above we can see an exact example of this statement. Phone numbers and addresses are the structure for a Personal contacts management application.

?? «XIS_enumeration»: is similar to the standard UML stereotype enumeration, but has some marked values to improve its capabilities.

?? «XIS_uiPart»: this class is based on the high level UIML concept of user interface part. This concept is wider than concepts as view, page or card once it allows a higher level of abstraction, allowing this way modeling systems with different types of user interfaces (e.g.: voice, ui for web, ui for wap).

?? «XIS_state»: state class represent a specific state of some XIS_entity object.

?? «XIS_action»: action class is used to describe possible actions in a particular user interface part or view (XIS_uiPart).

?? «XIS_map»: map class represents the transition between states originated by some action.

?? «XIS_application»: application class represents the whole applicat ion and based on this object we can model interactions between different applications.

**Association Stereotypes:**

?? «XIS_mustHave»,«XIS_isOf»,«XIS_contain»: these stereotypes represent types of possible associations between classes. The name of each one of these associations is illustrative of the semantics they represents

?? «XIS_navigate» - It is a special type of association that allows to specify navigability among several XIS_uiParts or to describe navigational behaviors inside each XIS_uipart.

The XIS vocabulary has many tags but we will highlight some that for its importance and direct mapping to the UML models, serves to better understand the process outline:

?? <entity> - The Entity element describes the entities of the application that are persistent. They correspond to the classes defined in UML, that the attribute persistence has the persistent value.

?? <view> - The View element describes application's groups of Entities views, for user interfaces, data de-normalization for processing, etc.

?? <action> - The Action element describes actions that can be executed on a data view by system actors.

?? <map> - The Map element describes the state transitions of a view. It corresponds to relationships between UML classes defined with the stereotype «state» and «action».

?? <enumeration> - The element Enumeration describes the domains you enumerated of the application. It corresponds to classes defined with the stereotype «enumeration» in UML.

?? <relation> - The relation element describes the relationships among the persistent entities of the application, that is to say, it

corresponds to the associations defined in UML between Entity classes.

# 7. CONCLUSIONS AND FUTURE WORK

The XIS project and related work will follow two parallel but complementary research lines. On one hand, the permanent actualization and study of the XML (XSLT and XMI) and the novelties emerging from the UML 2.0 specifications as well as all profiles and CASE tools with relevant impact on the subject of model driven software engineering.

On the other hand, we will push the development of information systems following this model-driven and generative programming approaches; in particular we will continue to research how to use and how to integrate the best practices in software architectures into the XIS project, namely into our suite of templates and generators.

# REFERENCES


[1] James Conallen, 2000. *Building Web Applications with UML*. Addison-Wesley.

[2] Alberto Rodrigues da Silva, Carlos Escaleira Videira, 2001. *UML, Metodologias e Ferramentas CASE*. Centro Atlântico (Portugal).

[3] David Carlson, 2001. Modeling XML Applications with UML: Practical e-Business Applications. Addison-Wesley.

[4] James Rumbaugh, Ivar Jacobson, Grady Brooch, 1999. *Unified Modelling Language Reference Manual*. Addison-Wesley.

[5] J. Craig Cleaveland, 2001. *Program Generators with XML and JAVA*. Prentice-Hall.

[6] RationalSoftware White Paper, 2001. *UML Profile for Data Modeling*. http://www.rational.com/media/whitepapers/Tp180.PDF

[7] David Carlson, 2000. Modeling the UDDI Schema with UML. Ontogenics.

[8] E. Nesime Tatbul, 2000. The Use of XML in Software Engineering.

[9] Gerardo Trinidad, 2000. *XML and Databases for Internet Applications*. Commonwealth Scientific and Industrial Research Organization - CSIRO). Australia.

[10] Booch, G., Christensen, M., Fuchs, M. and Koistenen, J. *UML for XML Schema Mapping Specification*. http://www.rational.com/media/uml/resources/media/uml_xmlschema33.pdf

[11] W3C. *Extensible Markup Language (XML) 1.0*. W3C XML Working Group. http://www.w3.org/TR/REC-xml

[12] W3C. *XML Schema Parts 0-2: [Primer,Structures, Datatypes]*. W3C XML Schema Working Group. http://www.w3.org/TR/xmlschema-0/, http://www.w3.org/TR/xmlschema-1/ , http://www.w3.org/TR/xmlschema-2/

[13] OMG. *Meta-Object Facility (MOF™) Version1.4*, OMG UML working Group, http://www.omg.org/cgi-bin/doc?formal/2002-04-03

[14] OMG. *XML Metadata Interchange (XMI®) Version1.2*, http://www.omg.org/cgi-bin/doc?formal/2002-01-01

[15] OMG. *Model Driven Architecture - A Technical Perspective*. OMG Architecture Board MDA Drafting Team, http://www.omg.org/cgi-bin/doc?ormsc/2001-07-01

[16] OMG. *UML 1.4 – chapter 2 – UML Semantics*. OMG UML Working Group, http://www.omg.org/cgi-bin/doc?formal/01-09-73.pdf

[17] OMG. "White Paper on the Profile mechanism", Version 1.0, OMG Document ad/99-04-07. OMG UML Working Group,

[18] Rational, 2000. UML Profile for Modeling Complex Real-Time Architectures. Rational Software.

[19] Gerd Wagner, 2002. A UML Profile for Agent-Oriented Modeling. Eindhoven Univ. of Technology.

Web:

[20] World Wide Web Consortium (W3C) http://www.w3c.org/

[21] Object Management Group (OMG) http://www.omg.org/

[22] XML.com http://www.xml.com/

[23] U2 Partners http://www.u2-partners.org/

[24] 2U Consortium http://www.2uworks.org/

[25] INESC-ID's ISG, The XIS Project. http://berlin.inesc-id.pt/projects/xis/