D.B.E.

Digital Business Ecosystem

Contract n° 507953

WP 18: DBE UML Profile

D18.2 UML for the DBE

Project funded by the European Community under the "Information Society Technology" Programme.

Contract Number: 507953

Project Acronym: DBE

Title: Digital Business Ecosystem

Version: 0.1

Deliverable N°: 18.2

Due date: 30th December 2006

Delivery Date: 6th May 2005

Short Description:
At the time being D18.2 presents an overview of the MDA, XMI and UML profile.

Partners owning: Soluta.net

Partners contributed: Soluta.net, SLE

Made available to: Internal Use Only

Versioning			
Version	Date	Name, organization	Description
0.1	1 June 2005	Pierfranco Ferronato, Soluta.net	

Quality check

1st Internal Reviewer: Miguel Vidal - SUN

2nd Internal Reviewer: Angelo Corallo - ISUFI

1. Table of Contents

2. List of Figures

3. Foreword

Deliverable D18.2 (Soluta.net in charge) depends on output from D18.1 (LSE in charge). In particular, D18.1 is supposed to derive formal constructs, algorithms, and data structures related to context-sensitive formal languages and building on automata theory. D18.1 made a premise that useful connections between automata theory and non-linear dynamical systems theory could be made during the first year of the project. These useful connections are envisioned to take inspiration from the relationship between DNA and cell metabolism to draw an analogy with the relationship between the language of BML models and the context in which the DBE services are instantiated to generate part or all of their code automatically: the dynamical systems and DNA was found to be more difficult than initially believed.

As a consequence, while D18.1 started to look at a wide range of issues underlying the connections between self-organising behaviour and formal languages, LSE are about 12 months late in achieving the results described above, which are expected to be reported in D18.4 (LSE, M24). This fact was communicated during the first DBE annual review and has been noted by the reviewers and the EC.

D18.2 is meant to build on these results to derive an UML profile. In the meantime D18.2 presents an overview of the MDA, XMI and UML profile.

4. The MDA approach

The Model Driven Architecture (MDA) approach to IT system specification focuses on the separation of the specification of system functionality (what the system is supposed to do) from the specification of the implementation of that functionality (how the specification is to be carried out) on a specific technology platform. It also defines means to transform independent system specifications to platform specific implementations. In this way the model traceability in the software development process[1] is maximized. The MDA is called "model driven" because it is model centric, all the system specifications are being expressed using models.

The MDA represents the natural next step in the continuous effort for abstraction. Historically speaking, this effort started with the evolution of the machine code to assembly language, from that to procedural languages, 3GL, structured programming, 4GL, object oriented, visual modelling, and ending for now with UML. Some people may remember the days when a C program was considered an independent platform specification of an application, where the platforms were the processors and the ways to transform the independent specifications to platform specific were the compilers. Of course, today the "compilers approach" is used everywhere in the IT industry and it is so common that it may pass unnoticed, but it worth to remember that was a time when this approach was an IT breakthrough.

The MDA is an architectural approach in which modelling languages are used as programming languages[2]. This can greatly improve the productivity and the quality of a project. As we said before, it is model driven in that it uses models to describe and control all the activities involved in defining, building, and maintaining a system. In Figure 1 - MDA Generation, below is represented one of the most important qualities of MDA, the possibility to automatically generate an application from a given model.

Mainly, there are two principal figures involved in the development process of MDA[3]:

- an MDA programmer. He/she receives the specifications from the functional modelling expert, which describe the processes, the data model, the services, the functionalities and the components of the system. The programmer MDA transforms the functional specifications in computational models, but without knowing inherent aspects of the technological specifications which will be adopted: the "programming" it said to "platform independent". The figure is in an intermediate position between a traditional programmer and a functional analyst, with the only important difference that the used language doesn't depend of any specific technology, it is not C++ or Java or C#, the primitives are linked only to the functional aspects, chosen from all its technical elements, by example Enterprise Java Bean, Web Service or .Net.

- a platform programmer. He/she knows the technical aspects, and often is a full team not only just one person, needed for the implementation of the software on the chosen platform, which can be Java EJB, Microsoft .Net, COBOL, web services or something else. These have to know a wide range of technologies as there are, for example transactions, security, messages, databases, networking, and tools like

1 In MDA terms this is called Model Driven Development.
2 This is a primer, we suggest reading [MDAGuide] for investigating MDA
3 MDD: Model Driven Development

application servers, RDBMS, IDE, FTP, ERM. Their purpose in a MDA environment is to describe in formal way how vary technologies concur to realise the requested functionalities. It's about writing "templates" which tell to the generator how to transform a general concept like "activity list", in a HTML table, a Java Servlet, an "Activity" class in Java, in an ER data model and/or SQL instructions for the management of the persistence.

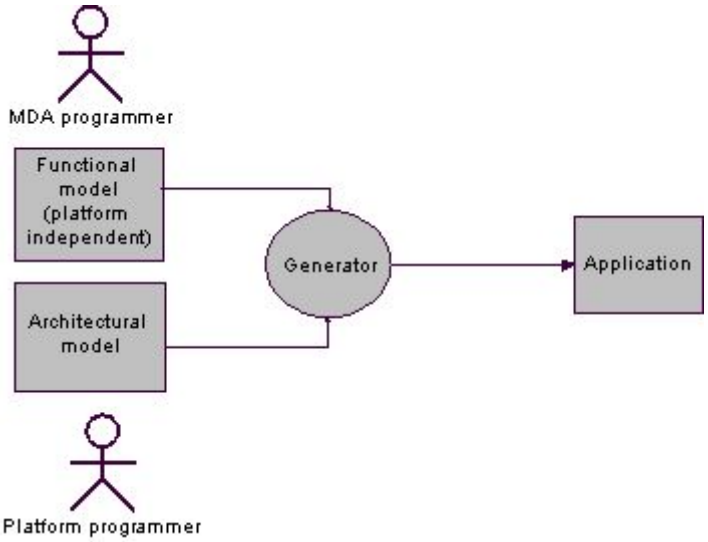

Figure 1 - MDA Generation

An important aspect for the work organization is that the programmer, from the moment when the rules of the application of the chosen platform technology have been defined, is not involved anymore in the iterations for development and maintenance of the project. To make a comparison, the platform programmer correspond to a developer of a C compiler; when the compiler is realized and the rules of the transformation in assembler have been implemented, it isn't necessary anymore involve him/she to write C programs of whatever nature.

MDA uses the MOF's (Meta Object Facility) concept of structuring the information in levels of abstraction. At each level of abstraction the information can be seen from one or more viewpoint and shaped into models accordingly. Three kinds of viewpoints are used, which lead to three types of models:

- a computational independent model (CIM)
- a platform independent model (PIM)
- a platform specific model (PSM)

In fact, the CIM and the PIM represent the specification of system functionalities and the PSM the specification of the implementation of this functionalities on a specific platform. The CIM focuses on catching the requirements of the system, i.e. what the system has to do, without saying how this will happen, while the PIM describes the implementation of the functionalities included in CIM in an platform independent manner.

The transformation of a PIM to a PSM is formalised by mappings created for each platform on which the system will function. If this transformation can be automated, a huge amount of time is saved, time used mostly to rewrite a system implementation, when its technical specifications have changed.

It can be said that the viewpoints and the views represent the MDA's horizontal approach to system modelling. All the models of a system, CIM, PIM, and PSM are constructed on the same level of abstraction. The vertical one consists in the four MOF's levels of abstraction.

The process of defining and implementing a system using MDA consists in capturing the system requirements into a CIM, implement these requirements as a functional but platform independent model (the PIM), and then chose one or more platforms on which the system should function and define mappings for the transformations of the PIM to the platform dedicated PSMs. Like this, if the business changes it should only update the PIM, and by the use of the defined mappings the PSMs are recreated.

MOF

The Meta Object Facility (MOF) is one of the key foundations of the MDA. It is a lightweight modelling language intended mostly to define other modelling languages which are dedicated to a specific domain, and it supports any kind of metadata that can be described using object modelling techniques.

In order to structure the information, MOF is using two dimensions. First, it defines four levels of abstraction. These levels can be seen as the vertical dimension because the information on one level is defined using information from the level above. A brief description of this levels can be found in Table 1 - MOF's levels of abstraction, below. Second, at one level of abstraction the information, i.e. the system to be modelled, is seen trough three viewpoints which leads to three types of views or models. Those are the CIM, the PIM, and the PSM. Because all three types of models represent data at the same abstraction level it can be said that this is the horizontal dimension of MOF.

Meta-level	Name	Description	Examples
M3	Meta-metamodel	It is the most abstract level. MOF is defined as being self describing.	The "MOF Model".
M2	Metamodel / meta-metadata	Here are defined the modelling languages which will be used for modelling.	UML Metamodel, CWM Metamodel, etc.
M1	Model / metadata	It consists in models which describe what an application deals with at a certain level of abstraction	UML Models, Warehouse Schemas, etc.
M0	Data	It contains the information described by the M1 level models	Modelled systems, Warehouse databases, etc.

Table 1 - MOF's levels of abstraction

One of the main purposes of MOF is to facilitate the interoperability between different metadata repositories and this is done mostly trough the use of XMI.

XML

The XML stands for eXtensible Markup Language and it is a language designed to describe, to carry, to store, and to exchange data.

Any XML file is text based, and its tags aren't predefined then they should be "invented". An XML file must be well-formed and it may be subject to a validation using a DTD (Data Type Definition) or an XML Schema. A well-formed XML document is a syntactically correct

one. A valid XML document is a document which has its content conform with a predefined DTD or XML Schema, which specify what an XML document may contain and how this content has to be structured.

When two different systems exchange data using XML based files or streams, first the data is encoded into an XML document by the sender, and then sent it to the other system. The receiver checks the XML document if it is well-formed (syntactically correct) and it validates the document against a predefined DTD or XML Schema before importing it.

XMI

The XMI or XML Metadata Interchange format is an XML based language used to interchange the objects of software systems. Those objects can be MOF/UML models, database schemas, program language classes and interfaces etc. Put it simply, any XMI document is in fact an XML document constructed following certain rules. These rules usually describe which are the elements that can appear in an XMI file, their multiplicity and structure. In the same way as the XML is a general solution for sharing data among different systems, the XMI is a particular solution for exchanging models. XMI is based on three industry standards – XML, UML, and MOF. It supplies rules by which an XML Schema can be generated for any valid XMI-transmissible MOF-based metamodel.

One of the most important uses of XMI is that it offers the possibility to have inter-product repositories. That can be done because it is used as a common form of representation of objects provided by different vendor tools which have only to be MOF/UML compliant. In this way the models are easily interchanged between different tools and even stored in an common repository. The benefits of such approach are invaluable, because the modellers aren't anymore constrained to use only one tool exposing themselves to become "captives" of the market strategy of the vendor who maintain that tool. A model being stored in an XMI MOF/UML complaint repository can be used by any tool who implement this approach, as we stated before.

An example of an XMI file is shown in Figure 2 - XMI example, at page 10. It is the XMI representation of the class diagram shown in Figure 5 - Using stereotypes, at page 13.

```
<?xml version='1.0' encoding='UTF-8' ?>
 <XMI xmlns:UML = "//org.omg/UML/1.3" xmi.version = "1.1">
 <XMI.header>
   <XMI.documentation>
   <XMI.description>Just an example</XMI.description>
   </XMI.documentation>
   <XMI.metamodel xmi.name = "UML" xmi.version = "1.3"/>
 </XMI.header>
 <XMI.content>
   <UML:Model name = "Example" xmi.id = "{66139940-B26B-4D68-8217-41D4BE5E3A4B}">
   <UML:Namespace.ownedElement>
   <UML:Class name = "Employer" isLeaf = "false" xmi.id = "{06D12C26-DADF-4137-A25A-25CFE9BB2855}" isAbstract = "false" visibility = "public"/>
   <UML:Class name = "Employee" isLeaf = "false" xmi.id = "{18878CBA-9257-429A-ADD2-CF7CE247AB31}" isAbstract = "false" visibility = "public"/>
   <UML:Association name = "Employement" xmi.id = "{15A94ACC-4CE6-410C-89E9-1873CF1EFEE8}">
    <UML:Association.connection>
    <UML:AssociationEnd name = "Employee" xmi.id = "{15A94ACC-4CE6-410C-89E9-1873CF1EFEE8}a" visibility = "public">
    <UML:AssociationEnd.aggregation xmi.value = "none"/>
    <UML:AssociationEnd.changeability xmi.value = "changeable"/>
    <UML:AssociationEnd.isNavigable xmi.value = "true"/>
    <UML:AssociationEnd.ordering xmi.value = "unordered"/>
    <UML:AssociationEnd.type>
     <UML:Classifier xmi.idref = "{18878CBA-9257-429A-ADD2-CF7CE247AB31}"/>
    </UML:AssociationEnd.type>
    <UML:AssociationEnd.multiplicity>
     <UML:Multiplicity>
      <UML:Multiplicity.range>
      <UML:MultiplicityRange>
       <UML:MultiplicityRange.lower>0</UML:MultiplicityRange.lower>
       <UML:MultiplicityRange.upper>-1</UML:MultiplicityRange.upper>
      </UML:MultiplicityRange>
      </UML:Multiplicity.range>
     </UML:Multiplicity>
    </UML:AssociationEnd.multiplicity>
    </UML:AssociationEnd>
    <UML:AssociationEnd name = "Employer" xmi.id = "{15A94ACC-4CE6-410C-89E9-1873CF1EFEE8}b" visibility = "public">
    <UML:AssociationEnd.aggregation xmi.value = "none"/>
    <UML:AssociationEnd.changeability xmi.value = "changeable"/>
    <UML:AssociationEnd.isNavigable xmi.value = "true"/>
    <UML:AssociationEnd.ordering xmi.value = "unordered"/>
    <UML:AssociationEnd.type>
     <UML:Classifier xmi.idref = "{06D12C26-DADF-4137-A25A-25CFE9BB2855}"/>
    </UML:AssociationEnd.type>
    <UML:AssociationEnd.multiplicity>
     <UML:Multiplicity>
      <UML:Multiplicity.range>
      <UML:MultiplicityRange>
       <UML:MultiplicityRange.lower>0</UML:MultiplicityRange.lower>
       <UML:MultiplicityRange.upper>-1</UML:MultiplicityRange.upper>
      </UML:MultiplicityRange>
      </UML:Multiplicity.range>
     </UML:Multiplicity>
    </UML:AssociationEnd.multiplicity>
    </UML:AssociationEnd>
    </UML:Association.connection>
   </UML:Association>
   </UML:Namespace.ownedElement>
   </UML:Model>
   <UML:Stereotype name = "person" xmi.id = "Strn0">
   <UML:Stereotype.baseClass xmi.value = "Class"/>
   <UML:Stereotype.extendedElement>
   <UML:ModelElement xmi.idref = "{06D12C26-DADF-4137-A25A-25CFE9BB2855}"/>
   <UML:ModelElement xmi.idref = "{18878CBA-9257-429A-ADD2-CF7CE247AB31}"/>
   </UML:Stereotype.extendedElement>
   </UML:Stereotype>
   <UML:TaggedValue tag = "persistence" value = "persistent" xmi.id = "TagVal0" modelElement = "{06D12C26-DADF-4137-A25A-25CFE9BB2855}"/>
   <UML:TaggedValue tag = "persistence" value = "persistent" xmi.id = "TagVal1" modelElement = "{18878CBA-9257-429A-ADD2-CF7CE247AB31}"/>
 </XMI.content>
 </XMI>
```

Figure 2 - XMI example

Extensibility

For object oriented modelling purposes the UML seems to be the perfect solution, but the IT universe doesn't contain only object oriented approaches, and the databases are a good example for that. For these reason instead to redefine the UML any time when a new domain needs it, and by that exposing it to become a huge and almost impossible to manage modelling language, the UML architects provided build-in extension mechanisms. These extension mechanisms make it possible to define additional modelling constructs further than the base UML provides. These UML-based languages created by extending the UML are called UML profiles. The UML profiles are also called the UML lightweight extension mechanism.

Another way to extend UML is at the metamodel level. A metamodel represents the structure and semantics of a set of models, and is in turn a model itself. The UML metamodel is expressed using MOF and an UML model is an instance of this UML metamodel.

Extending UML at the metamodel level involves the specialisation of the base UML metamodel for a specific domain, without changing the UML semantic or redefining any of its elements. The extension of UML by extending its metamodel is also called the UML heavyweight extension mechanism, mostly because it keeps all the base UML definitions while adding new ones[4]. The result of such extension process is an UML variant used to create models for a specific domain.

It has to be taken in consideration that any of the UML variants, obtained by the use of profiles or by specialising the UML metamodel, are specific languages and they cannot be universally understood and supported as the UML itself. The use of such variants is restricted to the organisation which implements them.

If a domain needs a dedicated modelling language then MOF can be used to create one who fits the needs of such domain. The MDA does not restrict the use of modelling languages only to UML. Another languages can be used as long their MOF metamodels are supplied. Such approach is necessary when a specific domain needs that a modelling language fits completely on its particular characteristics. As an example, the CWM (Common Warehouse Metamodel) is a modelling language used to create data warehousing models.

UML Profiles

Profiles may be defined for any language; in these pages we are referring to UML. An UML profile consists of a package that contains one ore more related extension mechanisms (e.g. stereotypes, tagged values and constraints). The extension mechanisms used by profiles are stereotypes and tagged values. A set of extensions, i.e. stereotypes and tagged values, which basically are an UML dialect is officially called a profile. In that way the UML is a foundation for a family of UML-based languages, instead being only one language.

Stereotypes

Stereotypes are used to extend UML entities in order to enrich the semantic of basic model elements that can hence be applied to specific domains. In fact, they are new classes of the UML metamodel defined at modelling time. It can be said that a stereotype is a refinement of an existing UML metamodel element. Because such meta-elements are defined at modelling time it gives flexibility to this method of extension. Usually the stereotypes represent different usages of an existing meta-element. These may also have required tagged values to complete the information needed by elements with the stereotype.

A stereotype can be represented as a name on existing UML element between «» (e.g. «InvoiceSystem»). An icon can be chosen to represent a stereotype, but such choice improves only the human readability of the model, without affecting its computational value (i.e. an icon cannot be taken as input by a generator). If multiple stereotypes are applied to the same model element the icon will be omitted.

4 For more details see [xx]

Metamodel class	Stereotype name	Example
Class	button	«button» OkButton; «button» CancelButton
Class	person	«person» employer; «person» employee
Class	database_table	«database_table» invoices

Table 2 - Examples of stereotypes

Some examples of stereotypes can be seen in Table 2 - Examples of stereotypes, above. One of them is represented in Figure 4 - Using generalization and Figure 5 - Using stereotypes, below. It is a simplistic situation that may not appear in the real world in this form, but when into an organization a concept seems to be fundamental it has no sense to repeat it in every model and for any class which refine it, but to "push" it into the following superior abstract level, the meta model, as a part of a profile which that organization uses.

Tagged values

A tagged value defines additional properties for any kind of model elements. It can be defined for an existing model element or even for a stereotype. In fact it is a tag-value pair where the tag represents the name of the new property and the value represents the value of the property.

For example a tagged value can be used to add information on an element of a model about code generation, version control, authorship, etc.

The representation of a tagged value is a string enclosed by brackets {} and which consists of a tag, a separator (the symbol =), and a value (e.g. {version=2.1.34, author="John Doe"} Invoice, «transactional» {autocommit=true}.

Constraints and comments

The constraints and the comments are other methods used to extend the UML (in general they can extend any modelling language).

The constraints are used to extend the semantics of UML by adding new rules, or modifying existing ones. These can also be used to specify conditions that must be kept true at all times for elements of a model. To represent the constraints the OCL (Object Constraint Language) is used.

The comments are used to improve the clarity of the UML models. They can be used to explain the logic behind some design choices. A comment is shown as a text within a note icon. Please bear in mind that comments are semantically poor, this means that they are not computable.

5. The MDA Generation Process

MDA identifies the existence of two types of models: platform independent (Platform Independent Model, PIM) and platform specific (Platform Specific Model, PSM)[5].

A simple MDA development process can be seen in Figure 3 - MDA Generation details below. There are then two phases in the generation process; from the functional model and the architectural model the structural model is generated and from this, in a second consecutive phase, the final platform specific application is generated.

The visibility of the structural model, a common feature not indispensable, contributes to make the generated model understandable and it serves as reference and documentation of the technical architecture. The functional model is written with a PIM model while the implementation model is made with a PSM.

The language used to describe both PIM and PSM models is the UML, which today is the

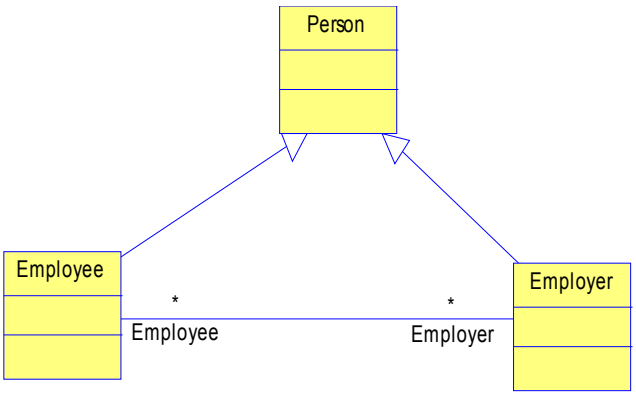

Figure 4 - Using generalization

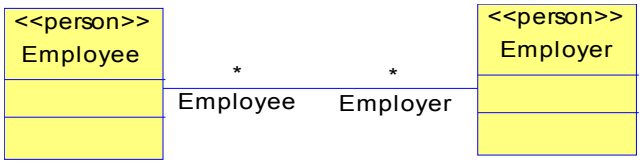

Figure 5 - Using stereotypes

only metamodel used by modeling tools producers. This allows in technical manner to use whatever modeling tool and for this, saving the investments for training and tools.

MDA Tools

The process that was briefly presented, is realizable by an MDA tool which can be developed independently following the OMG's specifications. In fact, the effort is considerable seeing the complexity of the specifications on which the metamodels and other profiling[6] and transformation standards are based. From years there are on market[7] MDA tools provided by big companies like IBM, SUN, Microsoft, and others which support the full software life cycle. An MDA solution has to be composed by three elements:

5 Another model type, not yet used in production environments is the Computational Independent Model (CIM)

1. an Editor (visual environment) for the creation of the models. The environment for model creation resembles to a graphical UML modeller. This modelling environment can allow, in some cases, the creation of a model using a modelling language (always PIM). Some MDA development tool doesn't provide their own model Editor[8], but embeds a third party, whilst another just imports the models[9].
2. one or more architectural models (see Figure 3 - MDA Generation details, page 13), called Cartridges, which describe the implementation technology and an architectural style. There are different Cartridges in a MDA tool; mainly there is one for any main platform, like are for example Enterprise Java Beans, Web Services and Microsoft . Net. Generally an MDA tool provides preconfigured Cartridges which can be part of the offer or purchased separately. It is possibly to exist even specific Cartridges for technological niches, like there are MQ Series, Java Messaging, COBOL, LDAP etc. In turn the user can modify the Cartridges or create new ones. It should be specified that an MDA tool in itself doesn't imply the selection of an execution platform for the final application; there isn't the risk to remain stuck on a provider or on a specific technical solution. A Java EJB Cartridge provided, can be substituted by another technology and the generated application will be in completely indistinguishable from one realized by a traditional approach.
3. a Generator (see Generator in Figure 3 - MDA Generation details, page 13). The Generator pretends as input the models produced by the previously presented tools (the functional model and the technological model), it maps them and generates the application. Like it was said before (see Figure 3 - MDA Generation details, page 13), an intermediary phase can be the generation of the platform model (PSM).

In relation with the roles described in at page , the MDA programmer creates the platform independent models using the UML editor, while the platform programmer writes the Cartridges.

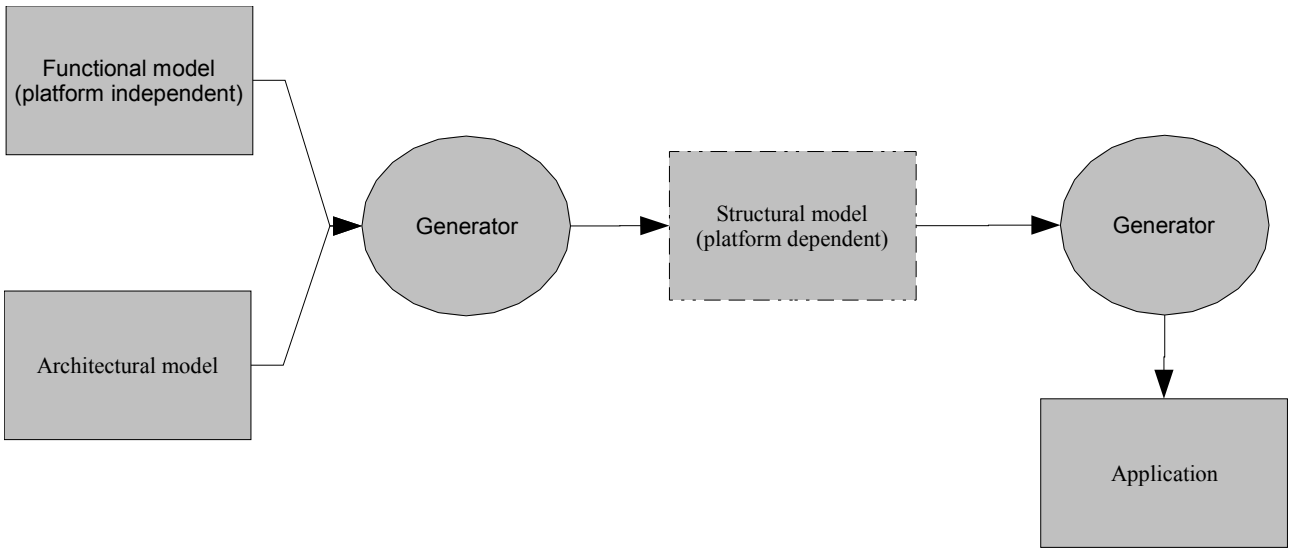

Figure 6 -Generation process

6 The profiling is an UML technique meant to extend the semantic of a model or metamodel. It is an approach based on the marking of the structures with tags and values and it doesn't require sophisticated extension techniques based on formal languages.

7 Demonstrating the mature state of MDA, there are also Open Source tools: AndroMDA, Eclipse EMF and NetBeans MDR.

8 ArchStyler from IOSoftware embed Magic Draw from NoMagic

9 Like AndroMDA. The import is made using an XMI file.

6. Bibliography

[MDAGuide]: OMG, “MDA Guide Version 1.0.1”, http://www.omg.org/docs/omg/03-06-01.pdf, 2003
[UML15]: OMG, “UML 1.5 Specification”, http://www.omg.org/uml

7. Glossary

Term	Description
ASP	Application Service Provider
b&b	Bed & Breakfast
B2B	Business to Business
B2C	Business to Consumer
Back End	In client/server distributed computing it is a generic term that refers to the runtime part of an information system that runs remotely with respect to the client hardware
Back Office	Department that has no or less contacts with the customers. From the wikipedia: "A back office is a part of most corporations where tasks dedicated to running the company itself take place."
Basic Service	Basic Services are DBE service that pertains to the following categories: payments, information carriers. The list could increase further on.
BE	Ref. Back End
BML	Business Modelling Languages
BO	Ref. Back Office
CIM	Computational Independent Model
CWM	Common Warehouse Metamodel
DBE	Digital Business Ecosystem
DNA	deoxyribonucleic acid
EAI	Enterprise Application Integration
EDI	Electronic Data Interchange
EDOC	Enterprise Distributed Object Component
FE	Ref. Front End
Front End	In client/server distributed computing it is a generic term that refers to the runtime part of an information system that runs locally with respect to the client hardware
GUI	A Graphical User Interface, it is a kind of UI
M0	MDA layer that references to M1 based data ('Mike Phone has invoice #1221')
M1	MDA layer that references to M2 based models ('customers may have invoices')

Term	Description
M2	MDA layer that references to MOF models (e.g. UML, CWM, EDOC, BML, SDL...)
M3	MDA layer that references to MOF language
MDA	Model Driven Architecture
MOF	Meta Object Facility
OMG	Object Management Group (www.omg.org)
P2P	Peer to peer
PIM	Platform Independent Model
PKI	Public Key Infrastructure
Proxy	Executable Java object that can be distributed over the Internet. It is usually a mediator to the actual remote service that provides the required functionality.
PSM	Platform Specific Model
RUP	IBM© Rational Unified Process®, or RUP®, http://www-306.ibm.com/software/awdtools/rup/
SDL	Service Definition Language: a language for the definition of a Platform Independent Model (PIM) of the service interface
Service Proxy	Ref. Proxy
SM	Ref. Service Manifest
Smart Proxy	Ref. Proxy
Super node	In P2P technology it represents a node of the network (a remote server). In the DBE there are several kind of super nodes: KB, FADA, DSS.
UDL	Universal Design Language
UI	User Interface
UML	Unified modelling Language
User Interface	Graphical User Interface
UUID	Universal Unique Identifier
VAS	Value Added Service: a service that is the aggregation of other services.
X509	It is a standard that makes it possible to identify someone or something on the Internet.
XMI	XML metadata Interchange
XML	Extensible Markup Language

Term	Description
XP	Extreme Programming http://www.extremeprogramming.org/
XUL	XML User Interface Language
ZIFA	Zachman Institute for Framework Advancement, http://www.zifa.com

--- end of document ---