Enterprise Model-Driven Development with BLU AGE™

By Franck BARBIER

BLU AGE™ J2EE .NET APPLICATION GENERATOR

NETFECTIVE Creative Technology

Enterprise Model-Driven Development with BLU AGE™
A Netfective Technology White Paper

by M. Franck BARBIER

Contact address

NETFECTIVE TECHNOLOGY
79, rue Jean-Jacques Rousseau 92150 Suresnes, France
www.netfective.com www.bluage.com
fbarbier@netfective.com

Content

I. MODEL-DRIVEN DEVELOPMENT

In the late '90's object-oriented modeling reached maturity. This lead to unifying OO methods in the Unified Modeling Language (UML®) [1-2]. Despite this notable progress, software production remained small-scale and the need for imagining "modern" industrial approaches, became evident. In the early 2000's, the Model Driven Architecture® or MDA® initiative [3] aimed at tackling this problem by laying down the foundations of model-centric development. This was mainly based on the idea of model transformation [3-4]. Other acronyms derived from MDA® appeared, especially those of MDE which stands for Model-Driven Engineering, and MDD, which means Model-Driven Development. The two previous acronyms are fairly free from the MDA® specification itself which, like UML®, is standardized by the Object Management Group™ (OMG™).

Beyond the deep technological nature of MDx (x = A, E or D), potential users who intend to investigate this innovative software development paradigm, obviously expect notable economical progress. This includes the ability to increase productivity due to modeling, a better fulfillment of time-to-market constraints by means of the easy and straightforward transformation of models into code and deployable applications. Taken as a whole, this corresponds to shorter returns on investments. However, there is no in-depth survey which actually proves that MDx increases profits in general.

In fact, while MDx benefits from having roots like object-orientation that induces reusability and maintainability, it is disadvantaged by the deep nature of modeling and models that are not accessible and understandable by average persons. This includes some software professionals. In this respect, the coexistence of MDx with other software development paradigms cannot be ignored, especially that of Rapid Application Development (RAD) which, from the beginning, promotes GUIs in applications as the first key entry point. Models, due to their abstract nature, may not be qualified as "natural" and "intuitive" for software developers. From experience, it is indeed difficult to convince them to replace code by models because they fail to have something concrete. Model executability is, in this scope, another way of facilitating MDx. Besides, model-based communication with non software specialists is tricky. Indeed, the latter prefer tangible software artifacts like GUIs to models. So, a smooth break is required to make MDx more appealing and efficient.

II. BLUEPRINTS TO ENTERPRISE MODEL-DRIVEN DEVELOPMENT

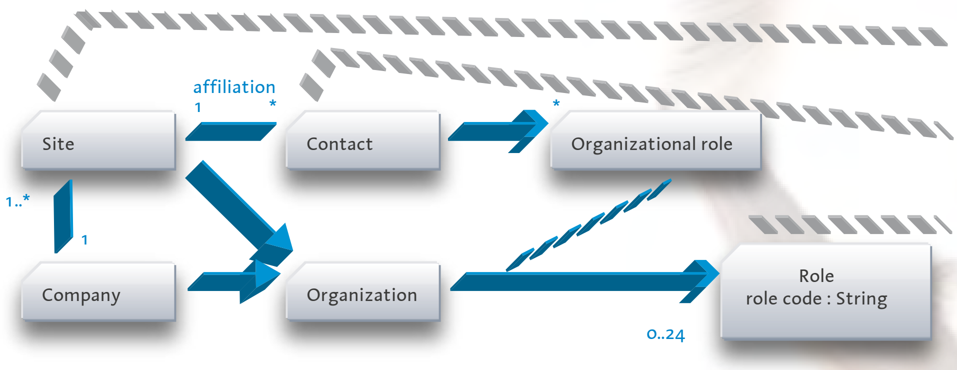
This White Paper is intended to demonstrate the necessity of moving from MDD to Enterprise Model-Driven Development (EMDD) from a conceptual viewpoint. From a practical perspective, it also describes an EMDD method which is made realistic through the use of two tools. MagicDraw™ (www.magicdraw.com) is a UML® modeler from the No Magic company. It fully supports UML® 2.x and XMI® [1] 2.x. Based on its capability plus its offering of an open API, MagicDraw™ enables the implementation of tailored MDD rules and processes. BLU AGE™ (www.bluage.com) is an J2EE .NET application generator from the Netfective Technology company. It relies on decorated XMI® models, which constitute its inputs. The decoration of models occurs through UML® stereotypes and tagged values predefined and accessible by means of a UML® profile provided by Netfective Technology. BLU AGE™ guarantees a 100% code generation, application packaging and deployment, provided that its coercive modeling method (the content of this White Paper) is punctiliously respected. This White Paper presents some screenshots coming from these two CASE tools. Like MagicDraw™ and BLU AGE™, all marks and brands cited and used in this White Paper remain the intellectual property of their authors.

1 XMI® is the standardized XML DTD for recording UML® models.

III. MODEL-DRIVEN DEVELOPMENT = NO CODE?

Advantages linked to MDD result from the foundation principles of software engineering in general and specification in particular. Models, through their abstract nature, favor early detection of problems, these being omissions, requirement misunderstandings and so on. Favoring does not however mean guarantying model consistency and completeness: all formalized requirements in a model are not contradictory, requirements are formalized so that they are neither ambiguous nor incomplete. So, even if MDD implies model checking, until now, there is no special focus on the built models' quality and more generally how model-centric software and information system development leverages productivity and speeds up time-to-market delivery. In fact, the key contribution of MDD is the definition of a rigorous model management framework rather than a precise, even new, modeling technology. To that extent, UML® is the preferred, but non exclusive, modeling language of MDD.

In the context of MDD, a Platform-Independent Model or PIM represents business requirements in a more or less advanced formal, graphical and/or textual form. For instance, a UML® Class Diagram plus some textual constraints (Figure 1) written with the Object Constraint Language or OCL [5], a UML® additive. Technical constraints often come next and MDD proposes the idea of a Platform-Specific Model or PSM, which is primarily a derivation of a PIM as shown in Figure 2 and secondarily, a description of a software system. For example, this system can be a software architecture omitting implementation details but referring to a special software technology. PSMs are the supports by which execution environment constraints may be first integrated and next satisfied seamlessly.Figure 1. A simple UML® model with its associated constraints



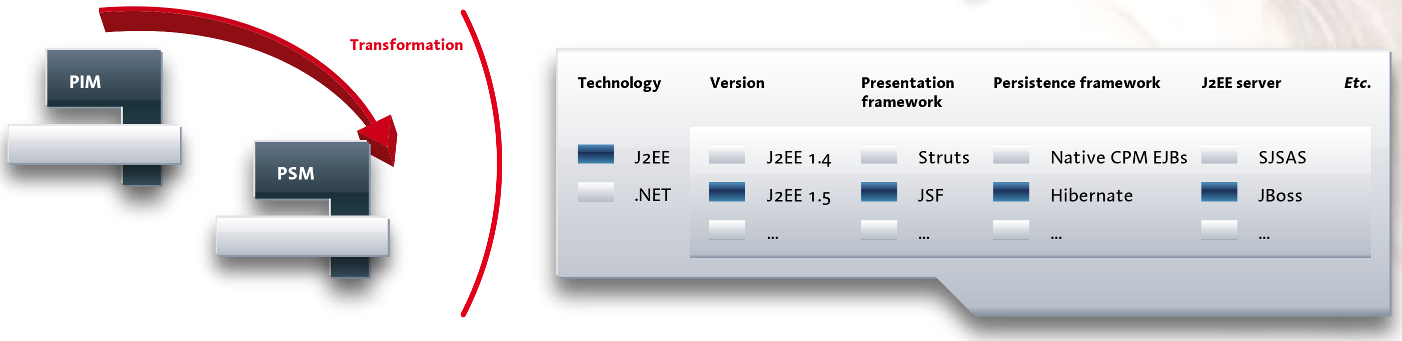
```
context Site inv:
-- the role of a given site (as an organization) are included in
-- the role set of the site's company.
company.role->includesAll(role)

context Contact inv:
-- the union of all roles of the site and the company of a
-- given contact's site includes the roles played by this
-- contact
(affiliation.company.role->union(affiliation.role)) ->
includesAll(organizational role.role)

context Role inv:
Role.allInstances()->isUnique (role code)
```

Figure 1. A simple UML® model with its associated constraints

In complex cases, several closely related PIMs may show how requirements are detailed and refined. The moving from PIMs to PSMs is also not so radical as shown in Figure 2. Several PSMs also exist, each results from lightweight or heavyweight choices like J2EE or .NET, the choice of a J2EE server, the choice of a RDBMS, the choice of a version of a J2EE server or a RDBMS, *etc*. The fact that technical constraint satisfactions are embodied in PSM appearances, demonstrates the ability of the model transformation rules and process to control and to trace model dependencies through code. MDD greatly and intensively emphasizes metamodeling, XML-based model and metamodel formats, open extensible model transformation languages and metalanguages and consequently, corresponding tools for managing models and their transformations (see [4] for a quite complete list of tools).



Technology	Version	Presentation framework	Persistence framework	J2EE server	*Etc.*
■ J2EE	□ J2EE 1.4	□ Struts	□ Native CPM EJBs	□ SJSAS	
□ .NET	■ J2EE 1.5	■ JSF	■ Hibernate	■ JBoss	
	□ ...	□ ...	□ ...	□ ...	

Figure 2. MDD spirit

Unfortunately, in MDD, the major attention is often rather devoted to model format management than model contents. From a semantic viewpoint, we mean that high-quality transformation rules cannot in any way transform a semantically poor PIM (a model whose content poorly represents the upstream requirements) into an adequate PSM, even if the latter is technically sound. In this scope, the table with options on the right hand side of Figure 2 governs a set of transformation rules which generate, more or less automatically, a PSM from a PIM. The modeling constructions which are allowed in the PIM, are processed in compliance with those accepted by the PSM; Notations at each level may indeed partially differ since modeling goals are different. However, ensuring that the PSM is an enhanced trustworthy image of the PIM (an isomorphism exists between the two), is not enough. The need for guarantying that the requirements formalized in PIMs are not altered during the transformation activity, is also important.

The main question is: To what point is the delivery of decorated models with platform-dependent features realistic? In other words, can we definitively believe that we can do it without writing code? In practice, a MDD process stops when the material in models does not enable the automatic generation of code. This means that applications appear in code templates or skeletons. Developers have thus to provide the very last implementation details and tuning. The difficulty is the control of such finishing touches to models, especially when these touches become sizeable. Experience shows that, for instance, performance improvement is often a cause of being at odds with the MDD spirit. In reality, it is thus difficult to avoid code writing, like for example providing glue code to be connected with specialized APIs. Psychologically, if software developers draw the conclusion that building models plus writing adaptation code is detrimental to productivity, there is a risk that they dismiss MDD. So, ideally, one expects from MDD the fact that no code is required. Since the models are rich enough, the adopted MDD rules and process enables the transformation of these models into complete and reliable code. Consequently, applications based on this code are easily and straightforwardly packaged for later deployment.

IV. CHARACTERIZATION OF ENTERPRISE MODEL-DRIVEN DEVELOPMENT

While MDD stresses model transformation concepts, techniques and tools, a limited focus has been put on modeling simplicity and intuition, as well as effective requirements validation through model validation. Validation goes beyond model checking in the sense that model reading and thus understanding is hard for average software developers, not to mention final users! For instance, an interesting application of MDD-based validation might be the generation of sentences in a weakened, scope-limited natural language. See for instance Figure 1 in which the OCL constraints benefit from being translated into English. Models expressed in a formal or semi-formal language like the UML® are indeed much too far from the way average users think, see and “touch” applications.

A key issue behind MDD is also model evolution management through requirements extensions and/or adaptations. No survey has really demonstrated that maintaining models is easier than maintaining code. This is a cause for concern in [6]. Moreover, the following question remains: How do MDD techniques and tools make upstream models (whose fluctuation cycles are those of requirements) perpetually consistent and inline with downstream models (which are subject to technological changes and improvements)? Besides, between PIMs and PSMs, several model layers may also exist and each intermediate model marks one of the different varied outputs of the chosen transformation rules and process. In short, while MDD is obviously a significant advance, it is not really clear how it addresses productivity issues, how returns on investment occurs, or what is the added value of MDD compared to competitors like RAD for instance. This global MDD assessment may be considered as rather pessimistic. However, from a different viewpoint [7], another demystification of MDD reinforces the vision of this White Paper. A breakthrough is in fact required: Enterprise Model-Driven Development.

Enterprise Model-Driven Development (EMDD) is the completion of MDD with well-proven software development technologies like RAD as well as the true integration of economical factors in a model-centric software development process. It is the building of maintainable, componentized (*e.g.*, SOA) applications, which support requirements engineering methods, especially through early requirements validation, and time-to-market delivery. Moreover, such applications obey to classical software quality concerns. The latter issue is notably addressed by MDD, which by definition complies with and applies all of the precepts of software engineering.

In short, EMDD = MDD + GUI-driven development + requirements management and adaptation that fulfill time-to-market demands. The latter point covers the idea of end-to-end management and adaptation from requirements capture to application generation, as well as test and deployment through model construction and maintenance.
EMDD may intentionally be compared to extreme modeling (the shadow of extreme programming) in which models have to lose their unappealing nature. Extreme modeling emphasizes model executability which corresponds to the simulation of models by means of instances and event occurrences, *etc.* If models are rich enough for 100% code generation, designers are therefore better convinced of the power of MDD. From a practical perspective, models may become direct (early) testing supports. This is the heart of the maintenance activity which consequently becomes less uncoupled (*i.e.*, not deferred) from the development activity itself. Different techniques may make models more worthwhile software artifacts for developers. As shown below, screens play a great role in such an approach. They form logical sequences of execution (in relation with models) and make models more tangible, especially from the users’ perspective.

EMDD is above all a pragmatic approach of software development. It is sketched in [8] and presented as “the reconciliation” of Activity Modeling and Usage-Centered Design. How EMDD precepts may be put into practice is now illustrated by means of an example and a proposal for an EMDD method.

V. BLU AGE™ EMDD METHOD

In this section, we emphasize the requirements engineering phaSE : this is an interview between a software engineer (SE) and the application’s buyer (AB). We illustrate the need for building high-quality PIMs before thinking about any PIM-to-PSM transformation. For that, one may remember the seven modeling sins formulated by B. Meyer in [9].

V.I Most Stories Rely on the Natural Language : A Case Study about the New York City Penitentiary

Below is an interview with the New York City Penitentiary (NYCP) director and the software engineer.

SE : For a given criminal case, can you have several prisoners?
AB : Yes, but we try to avoid it. I must also tell you that we include all the judicial decisions related to the incarceration in the prison file, such as:

- the convictions
- the shortened criminal sentences
- and the final discharges of the prisoner

Each of these judicial decisions is recorded in the prison file with their respective numbers (1, 2 or 3). Each decision has a date. The convictions include the length of imprisonment (in number of days) to be carried out. Shortened sentences include the amount of time the sentence is shortened by. The final discharge includes the date of discharge.
SE : For example, can a prisoner have multiple shortenings of his criminal sentence?
AB : Yes.
SE : You previously said that you registered the motive of the crime. Is it the motive for the conviction that the prisoner was sent to prison for?
AB : Yes, it is true that for the same crime there can be convicts condemned for different motives.
SE : Could you give me examples of some motives?
AB : Yes, for example :

- 01-theft and various misdemeanors
- 02-assault and battery
- 03-fraud
- 04-carrying of a weapon without a license
- 05-drunk driving
- 12-breach of trust
- 14-homicide
- 15-procuring for prostitutes, *etc.*

SE : For a given crime, there could possibly be several motives behind the conviction. Do you only record one?
AB : Yes, the main one.
SE : You also previously stated that you recorded the date when the criminal act was committed. However, is it possible that for a particular crime there may be several dates of crime? For example, a fraud may take place from a certain date to a certain date. So, for a specific crime, can there be several crime dates?
AB : We only record one crime date.
SE : Could you give me examples of some crime dates?
AB : Yes, for example, "during the month of April and May 1996" or "around December 6th, 1993".
SE : I would like to go back to the judicial decisions related to the incarceration. Can several judicial decisions (for example, shortened sentences) be registered on the same date for the same prisoner?
AB : No. I understand what you are saying. You think that one prisoner can receive two judicial decisions on the same date. For example two shortened sentences, of the same length, could be decided; each decision being linked to a different crime for which the prisoner is in prison. What you mean to ask is : Can two judgments, of the same kind, concerning the same prisoner, be registered on the same day?
SE : Yes.
AB : The answer is no. However, the judge can decide the same day to shorten the sentence or to condemn the convict to prison. It rarely occurs.
SE : Would you register the fact that a particular judgment, linked to the incarceration of a given prisoner was made by a certain jurisdiction?
AB : No. It does not matter to us which jurisdictions make judgments for these crimes.
SE : But what about the main crime, for which the prisoner is incarcerated, don't you record the original jurisdiction?
AB : Yes.
SE : Speaking of jurisdictions, can there be two of them with the same name?
AB : No.
SE : In your prison files, do you keep all the information on prisoners who have "legally" left the penitentiary?
AB : No. We only keep information on prisoners who are currently serving their prison sentence.
SE : Including prisoners under remand?
AB : Yes, of course.
SE : OK, I have got very interesting inputs about your information system. Now, what about your functional requirements?
AB : That is quite simple. I require the enforcement of law decisions, namely incarcerating a person. In such case, some information is created including a new prisoner record, its main criminal case, *etc.* The other kind of law decision to be enforced is when decisions are taken against existing prisoners, namely convictions, shortened sentences and final discharges. Finally, I must also be able to compute which prisoners are under remand. That is all for the moment.
SE : OK, thank you, I am going to build the UML® models.

V.II The Seven Sins of Modeling

The seven sins of modeling are, in alphabetic order: ambiguity, contradiction, forward reference, noise, over-specification, silence and wishful thinking [9]. Within the questionnaire above, each of these seven sins is illustrated. An evident expected outcome of MDD is how to better address requirements engineering issues; the presumed advantage of models compared to code. We show here that there is no miracle. An EMDD method must be grounded on efficient requirements engineering techniques which produce sin-free models. For that, no automated MDD process is nowadays a reasonable proven solution. So, requirements, as shown below, must still be managed "manually".

Ambiguity is characterized by an element in the requirements document that makes it possible to interpret a feature of the problem in at least two different ways. For instance, the prison director said: "However, an incarceration decision has obviously been taken against him." Later in the text, he talked about another kind of decision: "I must also tell you that we include all the judicial decisions related to the incarceration in the prison file, (…)". The ambiguity here relies on the fact that an incarceration decision, which is by definition unique for a given prisoner, is a completely distinct concept from decisions (*e.g.*, convictions) related to the prisoner's incarceration.

Contradiction covers the idea that two or more elements define a feature of the system in an incompatible way. In the requirements document, "the motive of the case" is recorded on each prison file. It represents the motive of the case for which a given prisoner was incarcerated. However, this prisoner may be involved in other criminal cases with probably different motives. Later, answering to a question, the penitentiary director said: "Yes, it is true that for the same crime there can be convicts condemned for different motives." He contradicts himself in the sense that he lets us suppose (or confirm?) that there are several motives for the same prisoner. So, do we have to manage, and thus record only one motive for a prisoner; That of the case for which he is incarcerated? Or, on the contrary, will we be able to know each motive linked to each case in which the prisoner is involved? We choose the first solution in the model appearing in Figure 3 (see also Section 6).

Forward reference refers to an element that uses features of the problem not defined until later in the text. A forward reference is not really a source of errors but disorients modelers in their thought process. For instance, the prison director introduces early in the conversation the important notion of "a prisoner under remand" while the triggering question was not about this point: "Can one prisoner enter the prison in relation with several criminal cases?" In fact, a prisoner under remand has no conviction decisions pronounced against him. This point has a direct tangible link with whether or not he is involved in zero, one or more than one cases.

Noise (redundancy may be considered as a subtype of noise) is observable through the presence in the text of an element that does not carry information relevant to any feature of the problem. Like forward references, noises cause interferences in the modeling thought and process. For instance, at a euphoric outbreak, the director said: "However, since I've been director of this prison, there have been no more prison breaks!". What is the relation with the information system the modeler is currently designing? None.

Over-specification is the presence in the text of an element that does not correspond to a feature of the problem but to features of a possible solution. The notion of "a prisoner under remand" is a possible source of over-specification. While this notion must be treated as a first-class requirement in the information system to be built, one must concomitantly anticipate that, as already said, a prisoner under remand has no conviction decisions pronounced against him. In terms of modeling, a bad idea is to materialize this notion as an entity. In the model appearing in Figure 3, a “zero-to-many” (*i.e.*, “*”) UML® multiplicity is used for modeling the notion of “a prisoner under remand”.

Silence is the worst sin. It is the existence of a feature of the problem that is not covered by any element of the text. How can one model what has never been said or written? This means that models are continuously changing software artifacts. In fact, the incompleteness of models is not only a mathematical notion but also a human one. Interview management through crosscutting questions for example, may help to transform silences into emerging requirements.
Wishful thinking corresponds to an element that defines a feature of the problem in such a way that a candidate solution cannot realistically be validated with respect to this feature. In the requirements text, the question “For a given criminal case, can you have several prisoners?” yields the response “Yes, but we try to avoid it.”. This answer is obviously a demand, but it contradicts reality.

V.III PIMs

The move from the text in Section 5.1 to the model in Figure 3 is not easy and straightforward, as seen in Section 5.2, traps are numerous. Independently of the quality of the UML® Class Diagram in Figure 3, which may be measured by the fact that requirements are fully captured and are not mutually contradictory, extra user-oriented validation is required.

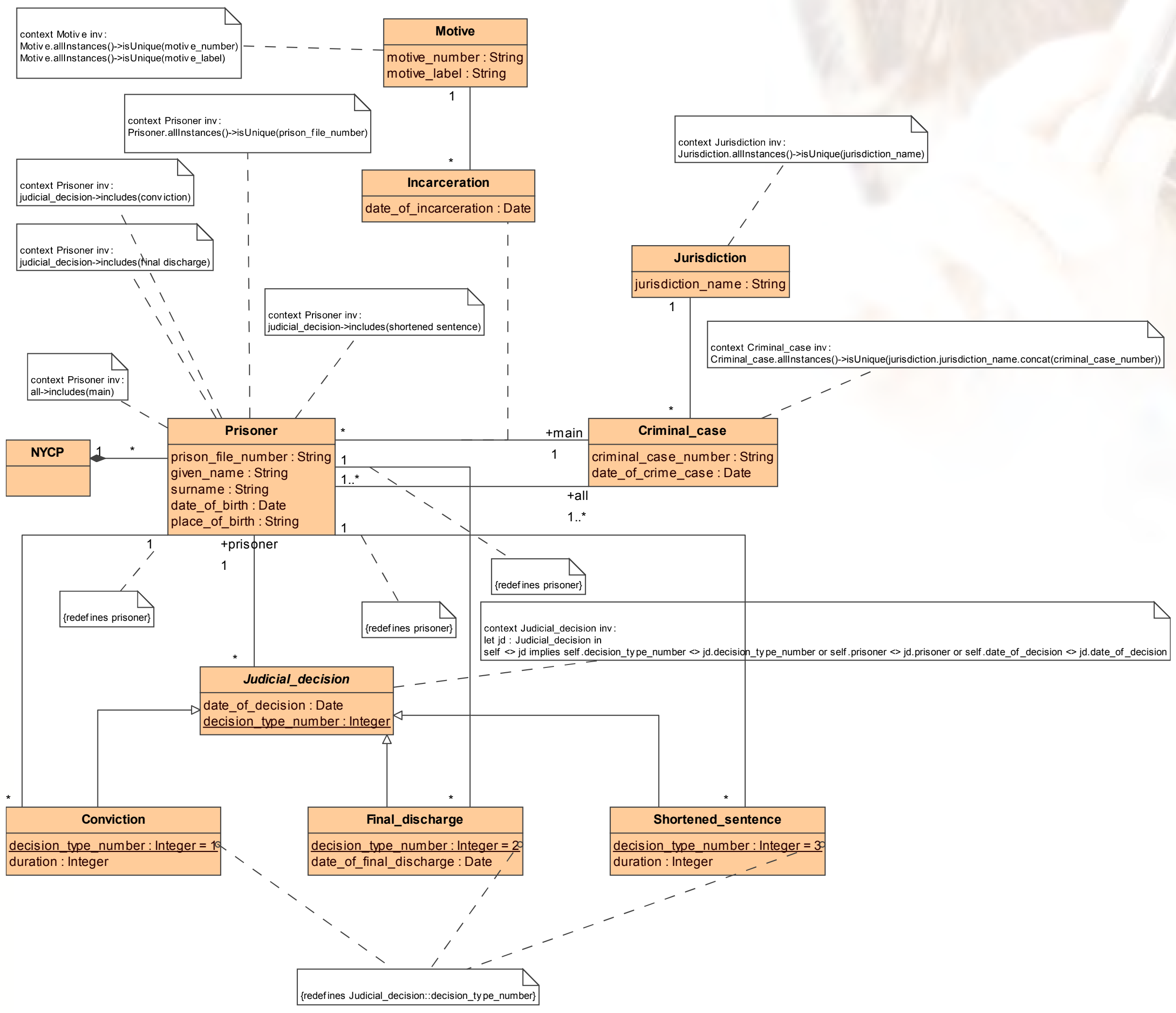

context Motive inv :
Motive.allInstances()->isUnique(motive_number)
Motive.allInstances()->isUnique(motive_label)
Motive
motive_number : String
motive_label : String
1
*
context Prisoner inv :
Prisoner.allInstances()->isUnique(prison_file_number)
context Prisoner inv :
judicial_decision->includes(conviction)
context Prisoner inv :
judicial_decision->includes(final discharge)
Incarceration
date_of_incarceration : Date
context Jurisdiction inv :
Jurisdiction.allInstances()->isUnique(jurisdiction_name)
Jurisdiction
jurisdiction_name : String
1
context Prisoner inv :
judicial_decision->includes(shortened sentence)
context Criminal_case inv :
Criminal_case.allInstances()->isUnique(jurisdiction.jurisdiction_name.concat(criminal_case_number))
context Prisoner inv :
all->includes(main)
*
NYCP
1
*
Prisoner
prison_file_number : String
given_name : String
surname : String
date_of_birth : Date
place_of_birth : String
*
1
1..*
1
+main
1
Criminal_case
criminal_case_number : String
date_of_crime_case : Date
+all
1..*
1
+prisoner
1
{redefines prisoner}
{redefines prisoner}
{redefines prisoner}
context Judicial_decision inv :
let jd : Judicial_decision in
self <> jd implies self.decision_type_number <> jd.decision_type_number or self.prisoner <> jd.prisoner or self.date_of_decision <> jd.date_of_decision
*
Judicial_decision
date_of_decision : Date
decision_type_number : Integer
*
Conviction
decision_type_number : Integer = 1
duration : Integer
*
Final_discharge
decision_type_number : Integer = 2
date_of_final_discharge : Date
*
Shortened_sentence
decision_type_number : Integer = 3
duration : Integer
{redefines Judicial_decision::decision_type_number}

Figure 3. PIM of the NYCP Information System

In this scope, Figure 4 complements Figure 3 with another PIM: Use cases.

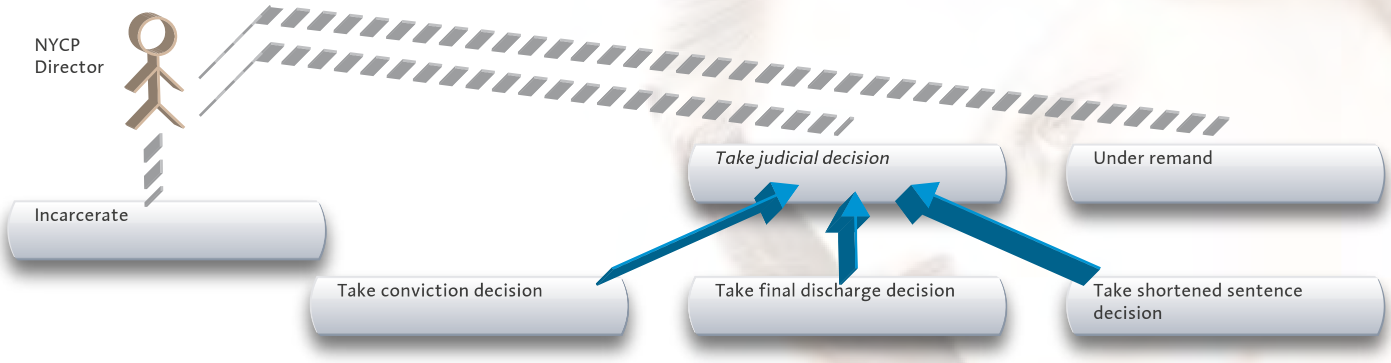


Figure 4. Use Cases of the NYCP Case Study

The missing link is how use cases interact with the information system in Figure 3. To solve this problem, additional models (*e.g.*, Activity Diagrams, Sequence Diagrams) are important and required. However, at this stage, MDD cannot be considered as an easily accessible discipline for end-users, or even developers. This is due to too much abstraction. Indeed, the adding of new diagrams at this stage increases the amount of abstraction. Here, abstraction is used in a pejorative sense, while it is recognized for leveraging software quality in general; This is an underlying principle behind MDD. In contrast, RAD (which attaches too much importance to GUIs) may favorably complement the PIMs in Figure 3 and in Figure 4 (Figure 5).

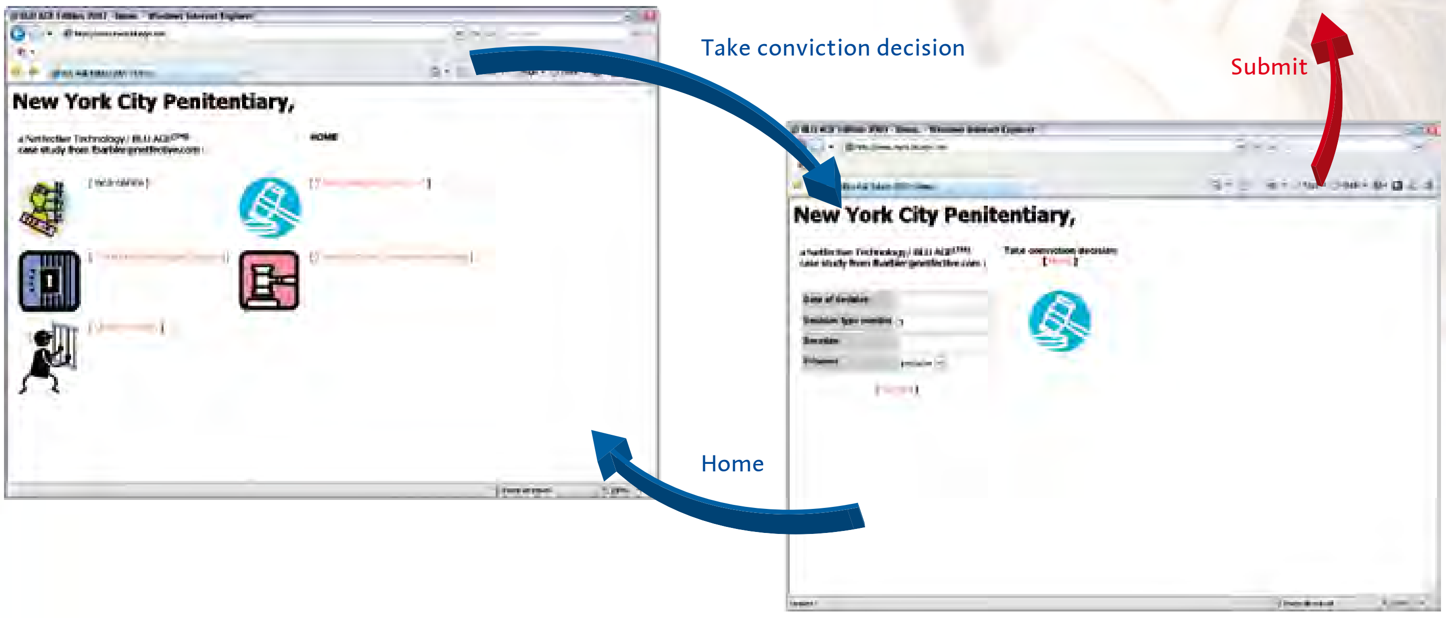


Figure 5. GUIs

In Figure 5, screen contents, dependencies and kinematics are valuable assets for end-users. Moreover, the screen on the left hand side of Figure 5 makes how the Take conviction decision use case in Figure 4 is operated at runtime more concrete. While the approach sketched in Figure 5 definitively has valuable properties, it is required to be well integrated with the models in Figure 3 and Figure 4.
Only under these conditions, may one benefit from both abstract and concrete material, each coming respectively from MDD and RAD. Moreover, the expected integration must occur, as far as possible, without code production (*i.e.*, under the auspices of a 100% application generation spirit).

V.IV EMDD Process

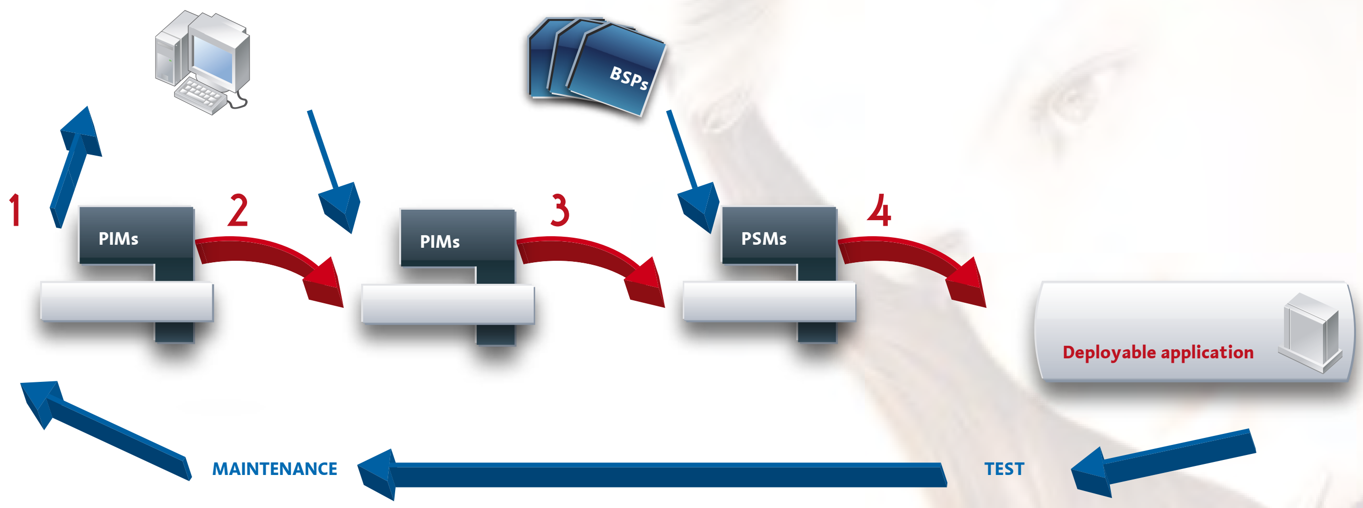


Figure 6. EMDD Process

So, the challenge is to create a gateway with objects in models and data in screens. The same goes for control flows in models, which map to logical sequences in screens. In the proposed EMDD process in Figure 6, screens are annotated with XMI®-like tags (phase 1, Figure 6).
For instance, the Submit button of the Take conviction decision screen in Figure 5 is assigned to an event of an Activity Diagram (Figure 8). In the same way, the displayed data is typed based on the types exposed in the Class Diagram extracted from Figure 3.
The first PIM-to-PIM transformation (phase 2, Figure 6) thus represents a step of a method that integrates MDD and RAD recipes.
The other key issue in Figure 6 is the handling of platform specificities by means of interchangeable models named BLU AGE™ Shared Plugins (BSPs). BSPs are indeed models themselves, expressed in the XMI® language. They refer to configuration data which is, for flexibility, externalized from the transformation engine. This enables the introduction of new platforms like proprietary platforms or the management of platform evolutions through news versions which are BSP variants. BSPs have to replace obsolete BSPs when applications evolve according to new technological targets. A metaphor of such a process is depicted in Figure 7.

Figure 7. A metaphor for the incorporation of a BSP into BLU AGE™

The third phase in Figure 6 is the instrumentation of what occurs in Figure 2. For instance, checking J2EE 5 instead of J2EE 1.4 amounts to integrating J2EE 5-specific properties in PSMs. The fourth and last step (phase 4, Figure 6) is the packaging of all the necessary files and libraries before deployment. For 80% of Web applications like the NYCP case study, the proposed EMDD method ensures that no code has to be written. In such a case, MDD really does make sense with respect to end-to-end development preoccupations, roundtrip engineering.

In Figure 8, screens (*e.g.*, home_screen and take_conviction_decision_screen, which both also appear in Figure 5) are modeled as states. The other states are called operations borne by business objects inheriting from the entities in Figure 3. So, state transitions match to screen sequences (take_conviction_decision event in Figure 8 which maps to the Take conviction decision hyperlink in Figure 5, left hand side), *etc.*

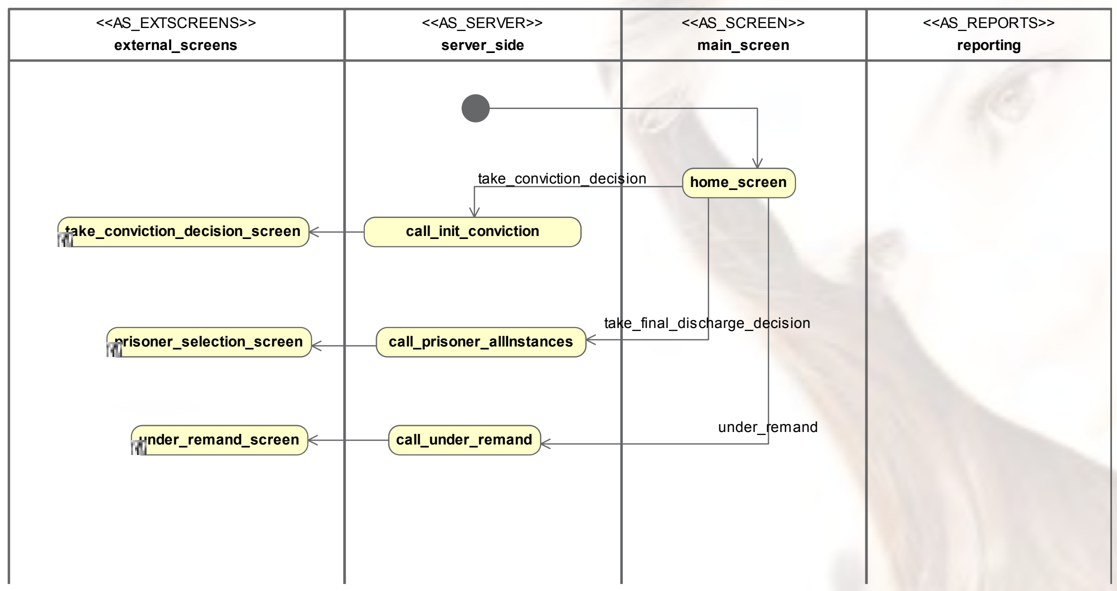
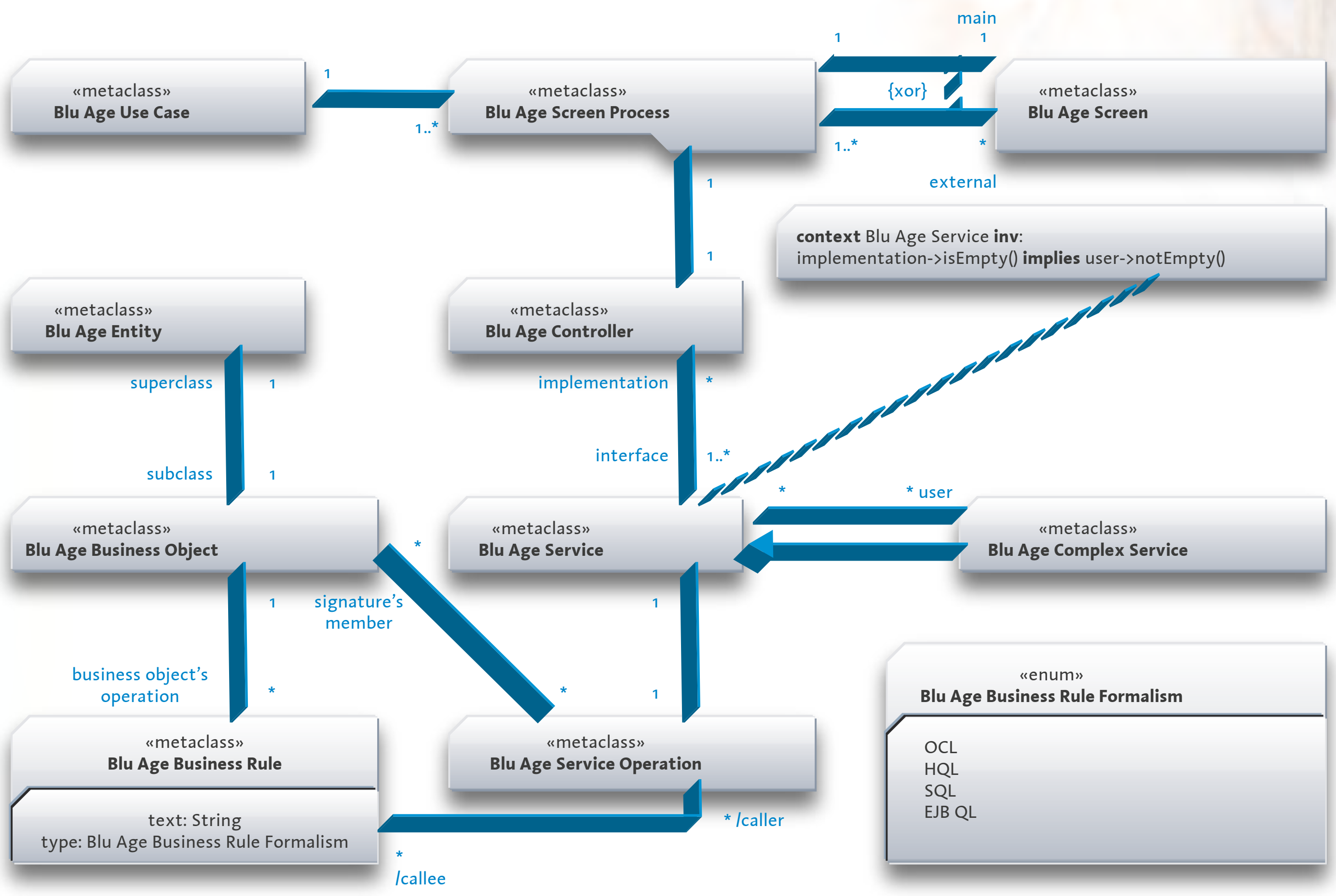


Figure 8. Screen kinematics modeling based on Activity Diagrams

V.V BLU AGE™ Metamodel

In fact, the proposed EMDD method is put against a coercive modeling framework appearing in Figure 9. For clarity, the modeling constructs required for constructing BSPs are not provided. Only constructs for modeling end-user applications are offered.



«metaclass» Blu Age Use Case
«metaclass» Blu Age Screen Process
«metaclass» Blu Age Screen
main
external
{xor}
context Blu Age Service inv: implementation->isEmpty() implies user->notEmpty()
«metaclass» Blu Age Entity
«metaclass» Blu Age Controller
superclass
subclass
implementation
interface
* user
«metaclass» Blu Age Business Object
«metaclass» Blu Age Service
«metaclass» Blu Age Complex Service
signature's member
business object's operation
«enum» Blu Age Business Rule Formalism
OCL
HQL
SQL
EJB QL
«metaclass» Blu Age Business Rule
text: String
type: Blu Age Business Rule Formalism
«metaclass» Blu Age Service Operation
* /caller
* /callee

Figure 9. BLU AGE™ Metamodel, a coercive modeling framework

Each BLU AGE™ metaclass depicted in Figure 9 inherits from an element of the UML® metamodel. For instance, the BLU AGE™ Service metaclass inherits from the Interface UML® metaclass. This first creates a compliance with UML® and makes modeling investments perennial. Next, instead of freely using UML®, a source of probable failure in [7], business models must be instances of the metamodel in Figure 9. The motivation behind this rule is the lack of guidance currently provided by the UML® and many associated tools. The move from MDD to EMDD indeed relies on a more disciplined and rigorous approach, which may be imposed by the coercive modeling framework in Figure 9.

VI. INCREMENTAL MAINTENANCE

One key expectation of MDD is the ability to update models in relation with new or adjusted clients' requirements. Such a maintenance has to occur based on a rapid cycle and in a cost-effective way. Heavyweight changes like, for instance, database restructuring generate high costs. As an illustration, the ability to know and to manage motives, not only for the main criminal case of a prisoner but for all criminal cases in which he is involved, imposes the modification of the Class Diagram in Figure 3: the UML® Association Class from Incarceration to the association (Prisoner, [main] Criminal Case) moves to the association (Prisoner, [all] Criminal Case). As for lightweight modifications, which represent around 90% of all of the maintenance cases, how may one prove that MDD actually reduces maintenance costs? A convincing response is required because, for instance, in [6], the author doubts the intrinsic claimed maintainability linked to MDD: "Model multiplicity mandates integrity maintenance among a system's various models, increasing exponentially with the number of diagrams. The recursive ripple effect of changing any model, thus triggering the potential need to modify other diagrams, renders intractable the problem of keeping coherent all system views." To sketch solutions for this tricky problem, we propose in the NYCP case study, the adding of some new functionality. In Figure 5 and in Figure 8, the Take shortened sentence decision use case captured in Figure 4, is not taken into account (the Take shortened sentence decision hyperlink is disactivated in Figure 5). Since the possibility to take shortened sentence decisions against prisoners is required in Section 5.1, we consider that the implementation of the Take shortened sentence decision creates the following maintenance tasks :

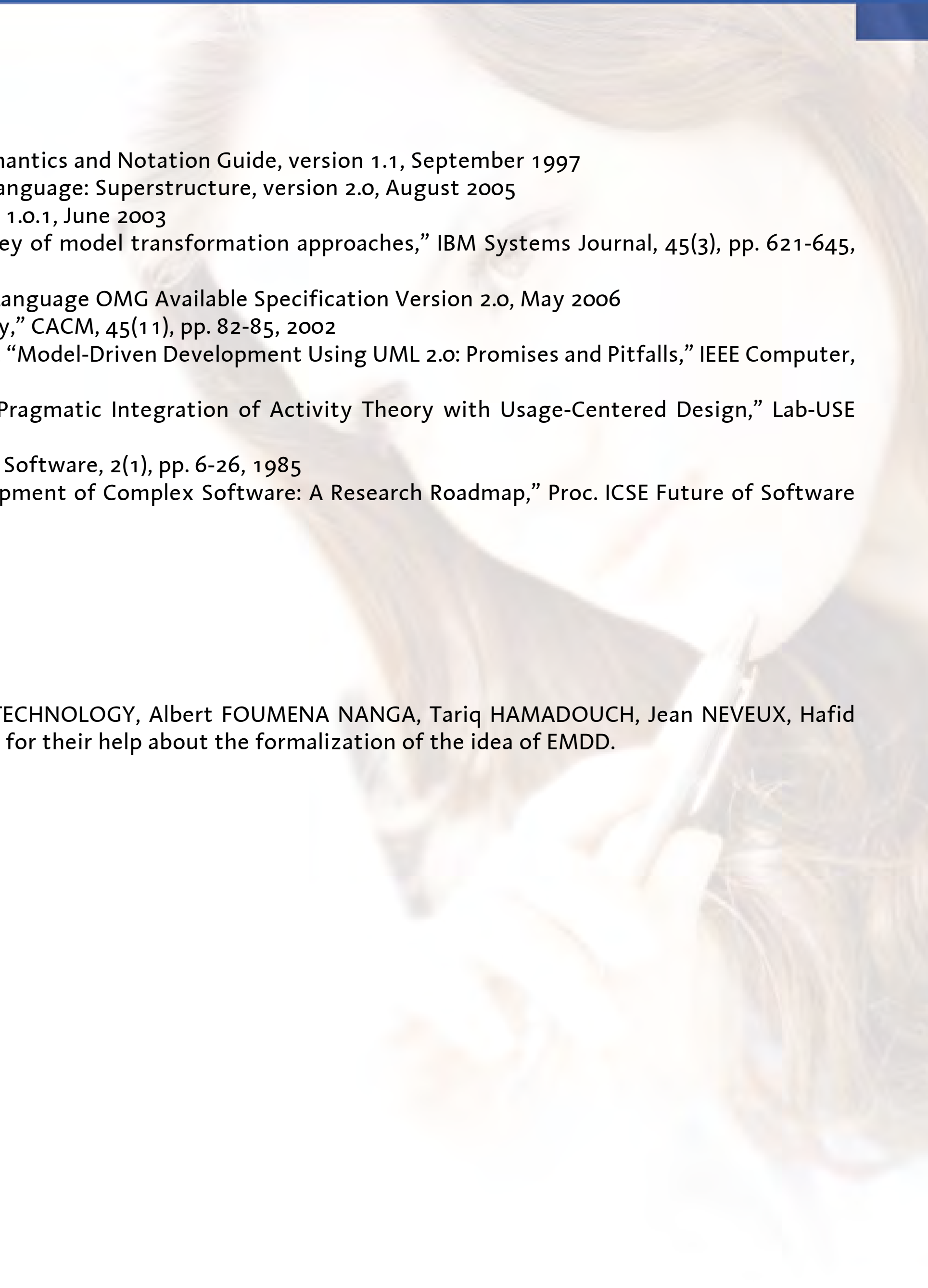
- design of a new screen, similar to that on the right hand side of Figure 5, which corresponds to the Take shortened sentence decision use case;
- extension of the Activity Diagram in Figure 8, via the introduction of the take_shortened_sentence_decision event between the home_screen state and a new state named take_shortened_sentence_decision_screen;
- design of an additional Activity Diagram for modeling the behavior of the application when the take_shortened_sentence_decision_screen is displayed and controlled, as well as other possible Activity Diagrams to enhance the current screens' sequence (renewed ergonomics, *etc.*)
- marking the inside of the new screen with XMI®-like tags in order to enable phase (1) in Figure 6.

When done, keeping in mind the hypothesis of no technological upgrade, the process in Figure 6 can be re-run without code handling at any stage of the maintenance cycle. Again, the quality of the model in Figure 3 is the key starting point (note that an association between Prisoner and Shortened sentence pre-exists and thus enables the previously described maintenance work).

VII. CONCLUSION

MDD is supposed to revolutionize the development of software through models which are, as far as possible, substituted for code. Code is nothing else than an operational model that includes all of the required details, which themselves relate to runtime platforms. Solving technical problems is however not enough. Engineers, developers and end-users must be convinced that MDD is productive, cost-effective, simple and intuitive. They may indeed stumble over too much abstraction and unfinished models, whose final touches are sizeable. EMDD aims at going beyond MDD by incorporating concrete material in the early phase of development, especially GUIs. EMDD also includes the possibility of really avoiding code through executable models and of 100% code generation. EMDD promotes the consistent integration of models and GUIs. It also emphasizes requirements engineering techniques that really take into account the famous seven modeling sins. More generally, MDD remains an open discipline since significant research challenges will exist in the future [10]. One of the three challenge categories in [10] is "model manipulation and management" as illustrated in this White Paper.

We show in this White Paper that EMDD is nowadays a reality through the MagicDraw™ and BLU AGE™ CASE tools. MagicDraw™ makes the approach offered in this White Paper fully compliant with usual standards (UML® 2.x, OCL 2.x, XMI® 2.x and MDA®) while BLU AGE™ completes the modeling phase with the incorporation the XML-based formalization of GUIs into business models and the flexible management of platform particularities and variations via BSPs. By means of an EMDD method, models truly constitute the best support for business knowledge capitalization since code is automatically and fully generated. This code remains the intellectual property of the company practicing EMDD. In any case, it may be directly modified even if one may consider that as an inappropriate idea. More interestingly, changing BSPs is the safer way for changing its nature and form (Java vs C#, JSF vs Struts APIs, SQL92 vs SQL99, *etc.*).



BIBLIOGRAPHY

[1] Object Management Group, UML Summary, Semantics and Notation Guide, version 1.1, September 1997
[2] Object Management Group, Unified Modeling Language: Superstructure, version 2.0, August 2005
[3] Object Management Group, MDA Guide Version 1.0.1, June 2003
[4] K. Czarnecki and S. Helsen, "Feature-based survey of model transformation approaches," IBM Systems Journal, 45(3), pp. 621-645, 2006
[5] Object Management Group, Object Constraint Language OMG Available Specification Version 2.0, May 2006
[6] D. Dori, "Why Significant UML Change is Unlikely," CACM, 45(11), pp. 82-85, 2002
[7] R. France, S. Ghosh, T. Dinh-Trong and A. Solberg, "Model-Driven Development Using UML 2.0: Promises and Pitfalls," IEEE Computer, 39(2), pp. 59-66, 2006
[8] L. Constantine, "Activity Modeling: Toward a Pragmatic Integration of Activity Theory with Usage-Centered Design," Lab-USE Technical Paper, Draft – Revision 2.0, 2006
[9] B. Meyer, "On Formalism in Specifications," IEEE Software, 2(1), pp. 6-26, 1985
[10] R. France and B. Rumpe, "Model-driven Development of Complex Software: A Research Roadmap," Proc. ICSE Future of Software Engineering, May 23-25, 2007

ACKNOWLEDGMENTS

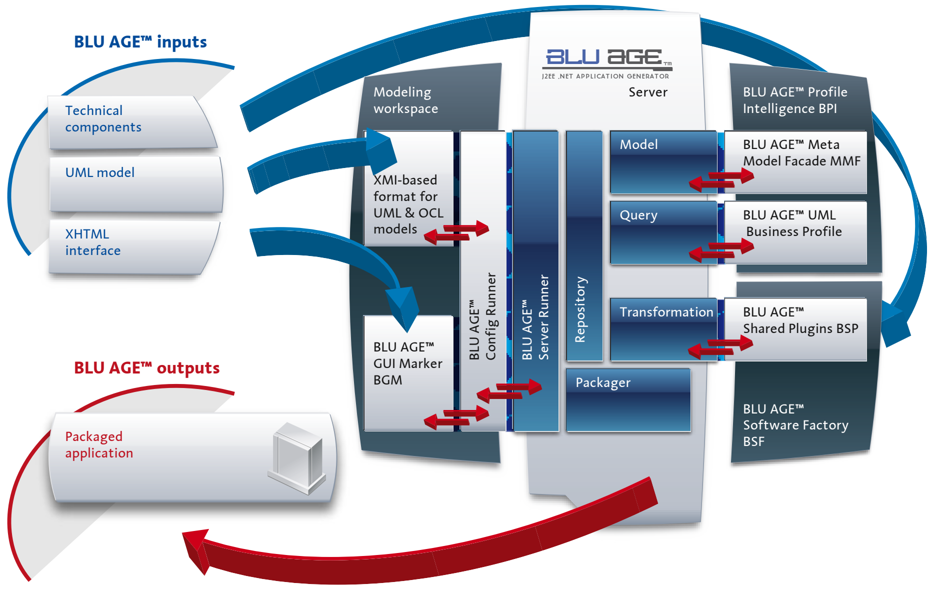
I wish to thank my colleagues from NETFECTIVE TECHNOLOGY, Albert FOUMENA NANGA, Tariq HAMADOUCH, Jean NEVEUX, Hafid OUAHIDI, Frédéric VERMENOUZE and Kamal YOUBI, for their help about the formalization of the idea of EMDD.

THE BLU AGE™ GENERATOR

BLU AGE™, the application generator built by the NETFECTIVE TECHNOLOGY R&D department is a pragmatic implementation of the MDA® (Model Driven Architecture®) standard developed by the OMG™ (Object Management Group™).

BLU AGE™ allows companies to shorten their development cycles, to reduce their implementation costs. BLU AGE™ also decreases the risks inherent to requirements engineering by means of a rigorous modeling method.

Based on BLU AGE™, Business Process modeling, implementation and deployment really create added value and quality in companies' information systems.



CONTACT

contact@bluage.com

NETFECTIVE TECHNOLOGY is an OMG™ member.

NETFECTIVE supports

PLANET FINANCE
www.planetfinance.org
The Microfinance Platform

QR Code www.bluage.com

PARTNERS

vmware® premier PARTNER