July 2013

Healthcare Membership Software Modernization

Successful model-driven modernization is teaching us humility

WHITE PAPER

Content

Pictures : http://autoautomobiles.narod.ru

1 Introduction

Miracles do not exist in Information Technology (IT). While (crazy?) evangelists invite companies to port applications to the Cloud, people maintaining legacy applications still daily wonder how these applications can deliver to clients, partners, members, colleagues… the expected services in time and with quality. From a business and cultural perspective, these individuals love a job well done. From a computer perspective, they have the feeling that they can satisfy a proliferation of constraints. Are they prehistoric human beings in a world of intelligent, connected robots? Cloud people asked them. Accordingly, they thought hard. Next, they found a path: they follow the blu prints for software modernization. The "blu" prints, not the blueprints, for the "blu" age. The age of maturity has come up; it is time for revamping their legacy information system. Of course, they had no actual choice, but they proved that they were superior to robots. That is a shared story between BLU AGE CORPORATION and one of its customers in the Healthcare Membership sector; that is also a humble and respectable vision on legacy information systems and people in this area, how these old systems may fit in the current software technology race.

Information is at the heart of the economical war. No company like this actor in the so-competitive Healthcare Membership sector may survive (often brutal) technologies' evolutions. Costs, costs, costs and costs, everybody knows this song. Spending money in maintaining dinosaur applications is no longer a solution. Spending money to experience the Cloud or any buzzword-like high tech is not a credible alternative especially when promoters only talk in rough terms. Risks are not under control. Instead, a smooth approach is the deal. For people in the "legacy area", thinking about psychotherapeutic analyses leads them to learn that the solution is to make their strengths emergent and to build on the top of these strengths, to capitalize on them. Assistance is a key additional ingredient. Its name is Blu Age®.

The key ideas:

- In companies, people coping with legacy applications are developers, architects, business analysts; they are like vintage vine. Daily hard labor, lacking time, have prevented them from mastering the contemporary technologies. However, how deep their knowledge of their company's requirements is, how high is their know-how, how meticulous is their care of the company's information system, how high is their colleagues' trust… creating an immeasurable value. The software modernization project sketched in this paper wants to take advantage all of these companies' features. **Software modernization cannot then be summarized in rejecting a "technological" information system. It is rather extracting gold nuggets with the notable help of very experienced miners**.
- Legacy infrastructures are often suitable and optimized; they are like tree ecosystems making up forests. Trees are elements of natural beauty when leaves fall down in multiple autumn colors. Nonetheless, nature is such that species vanish. In IT, legacy technologies pose a significant risk at the time they are no longer supported by vendors. Information and its associated computing processes living on the top of these infrastructures have reached an inestimable value in the sense processes perfectly represent and implement working

methods. Processes' value in particular comes from the energy spent to design them (structuring, tuning…). So, how to put aside legacy infrastructures while keeping the natural beauty? In other words, how to plant and make sustainable beautiful forests? This project is wary of shaking up the existing information ecosystem in an inappropriate way. In business sectors like the Healthcare Membership sector, destabilizing established organizations must not occur only because there is an urgent need of installing a new information system. **The transparency of the ways information is processed and pushed in users' hands is crucial, keeping, as much as possible, working methods and organizational practices stable, efficient, even more innovative**.

- Up-to-date technologies have (too much?) natural sex-appeal. At the time COBOL was the newest technology, people encountered definite setbacks due to its lack of maturity. It took long years before people were able to first tame and next fully control COBOL information systems. In this project, COBOL Pacbase (a COBOL dialect from IBM) is the legacy COBOL of interest. Its key characteristic is being well-structured because COBOL Pacbase is a kind of macro-language favoring the production, through code generation, of code bases in repositories. Despite this great advantage, code volumes, homemade adaptations, local (more or less "best") practices… have generated an information system whose intelligibility becomes, in time, lower and lower. From an external viewpoint, rationality is often absent although the overall information seems under control. Anyway, the situation is unsustainable in the long term. Changes are required including the switch to newer (Cloud-like) technologies making information systems more reactive and manageable. However, who can prove that porting applications to the Cloud is not the creation of, at least, the equivalent legacy information systems of the future, a nightmare IT world? In general, cloud infrastructures, platforms and services often appear as five-star jailhouses. Fortunately, nowadays, people learned from "models" to not redo the same mistakes. Roughly speaking, **one must definitely reject any adherence between, on one side, infrastructures and platforms (computers, networks, operating systems, middleware, APIs), and, on the other side, information and programs. Let us look at model-driven development**.

So, people in the legacy area devote themselves entirely to their information system because it is the crux of the matter in the economical war. Like competitors, cutting costs, satisfying clients, convincing internally are everyday challenges. When technology advances, this does not lead these people to radically transform their information system to meet these challenges. They modernize it, permanently maintaining, even increasing, the value of the information, going to the future with serenity, even towards the Cloud, in an informed consent.

You are going to read a story between BLU AGE CORPORATION and a major Healthcare Membership payer in the US market. This story gives us more insight into information system modernization by means of model-driven development. This is a happy story but it is above all sweat, hard labor, sincerity, real-world business life. No magic, no miracle.

2 General note

This white paper is a free dissertation based on, and inspired by, a more detailed white paper [Selip, 2013], which narrates the Healthcare Membership Software Modernization with accurate facts, numbers and many other informing elements. This white paper is a readers' digest of the former. It is an entry point to model-driven software modernization, towards the Cloud in particular.

3 The case study

This white paper is a post-analysis and reflection on a software modernization project. It is about lessons learnt from an ongoing large-scale case study in the US market (20 million COBOL Lines of Code). It is the modernization of a business-critical information system.

The first isolated piece of this information system is made up of 30 green-character screens, 500 standalone (online or batch) programs (25% of the overall system functionalities) and 400 data tables.

The project, which started in January 2011, released in August 2012 the renovated version of the first piece to be implemented in production. This encompasses the service-oriented representation of the legacy piece's functionalities as UML platform-neutral models. This also covers the full generation of code from these models to implement and run these functionalities within the targeted architecture and its underlying platform.

As an "almost ordinary" software project, many technical issues were addressed including for instance testing: system integration testing, user acceptance testing, etc. Practically, it was above all a huge software development shipyard, but this Herculean job had many other fascinating facets: the meeting of the cultural background of the customer's and provider's teams, the resulting collaborative work on novel technologies like model-driven development, the on-the-fly definition of an innovative software modernization engineering method… Indeed, beyond facts, numbers… (see [Selip, 2013]), it was a human adventure in a coercive economical context:

- Carrying out the project in "reasonable" costs,
- Anticipating users' large participation and adoption,
- Respecting and preserving professional culture in a so-specific domain (Healthcare Membership),
- Stressing a seamless adaptation to daily practices and operating methods.

In such a context, software modernization should be an odyssey.

In this case study, going on with the same software was no longer an option because of the impending suppression in 2015[1] of the vendor support of the legacy technology. Of course, this vendor offered a substituting technology not so far from the legacy one in the sense that COBOL remains the core programming language. Despite some security, even comfort, brought out by this offer, the guidance was to move out of COBOL in general and COBOL Pacbase in particular. This pretty revolutionary action came from the following non-negotiable requirements:

[1] Support stop was announced in 2007 to envisage and start preparing software evolution as soon as possible.

- The solution shall provide a tiered, rather than a monolithic (legacy style) architecture,
- The solution shall not be based on proprietary technology,
- The replacement effort shall minimize disruption to the business,
- The solution shall minimize cost of the entire effort on a lifecycle basis.

As a result, the central challenge was to put aside a very special computing environment, namely COBOL Pacbase, to the benefit of a "modern" one. From the beginning, it was not a one-shot operation: typically, code transcription. In other words, the solution cannot rely on a rigid-to-rigid, or monolithic-to-monolithic, mapping.

Indeed, in a world of permanent zapping from yesterday-new-today-old products to fashionable consumables, software has a surprising (damaging) feature: its malleability is dramatically poor, even non-existent.

The system to be constructed cannot then consist merely in transcribing. The risk is the transcription of useless statements or blocks whose existence is only linked to the constraints imposed by the legacy platform. Redeveloping the overall system was the other (extreme) alternative with, among many other dangers and probable penalties, the risk of uncontrollable costs.

In short, many strategies were considered with associated solution providers including, of course, Blu Age® Reverse Modeling and Blu Age® Forward Engineering. The choice of Blu Age® was mainly decided from the seeing-is-believing principle. It is a Proof-of-Concept (PoC), *i.e.*, a Java software prototype using the latest Web technologies and implementing a well-delimited part of the legacy information system. The PoC convinced people of what follows:

- "As-is" legacy architecture can be consumed and analyzed,
- To-be modernized architecture will be produced as expected and desired,
- Generated code quality is in line with the customer's expectation and standards,
- Generated code is free from any Blu Age® proprietary runtime software and is built on industry leading open source software technologies and frameworks, namely the ASTM, KDM and UML standards published by the OMG®, the Eclipse Modeling Framework (EMF) as well; EMF is a *de facto* standard in the Eclipse IDE (Interactive Development Environment),
- Business and technical risks of the project are uncovered, evaluated, and addressed,
- Velocity of modernization is determined for workload estimation and project planning.

Blu Age® handles "models" only. In this project, by strongly considering "models", it was the consolidation and the definition of a software modernization engineering method with an *ad hoc* process and an associated dedicated tool for maximal automation. Of course, "press on the modernize button" was not the dreamed situation. The human experience was the meeting between, on the first side, experienced people close as possible to their business, their information system and, on the other side, Blu Age® specialists always ready to investigate the latest technologies including model-driven development. Humility was the rule for everybody. A software application without business soul is not a service. It is just "technology for technology", something useless and costly. Today, the model-driven engineering approach validated in this project is not only supported in the Blu Age® software tool suite. It is the effective background of BLU AGE CORPORATION and its

customer in the Healthcare Membership sector. This background is gold. Customer's stakeholders were the fine connoisseurs of the mine. Blu Age® was the revolutionary extracting tool. Blu Age® specialists learn a lot about the mine when helping stakeholders mining. The modernized application is like a jewel.

4 Software modernization

In the IT world, years have delivered billions of lines of code. Years have also instilled complexity, complexity upon complexity… This comes from repeated demands from end users. Adjustments, adaptations, extensions, customizations… over years have drastically changed software shapes into formless architectures. It is like a car category without a tangible chassis, ordinary engine… The strange thing is that drivers are satisfied and mechanics are almost able to make these cars evolve. Nonetheless, car parts are no longer available or they must be fabricated in a tailored way; mechanics are few, have high salaries and would like to retire; traffic rules evolve and become incompatible for this car category, obsolescence problems raise on an exponential scale.

In this car analogy, mechanics are software maintainers. It is impressive the accumulated know-how of these people. In this Healthcare Membership company , abandoning a legacy information system is not only an era switch, it is a business-critical action, which determines the economical future, the company's survival. After years, business depends in an increasing way on the information system. From another perspective, recasting an existing information system is a strong opportunity to boost the business.

In this context, software modernization is a strongly demarcated alternative to:

- Developing a new information system from scratch, *i.e.*, from none or from general-purpose Component-Off-the-Shelf (COTS) components, even fully packaged solutions. This approach was rejected early because of business disruption. In other words, such approaches do not re-explore and capitalize, in a systematic way, all the value and huge labor engraved in the legacy information system. The data semantics and how it is processed in the old system are finely cut diamonds. They are lost in an electronic circuit, which resembles a spaghetti graph. This comes from the software architecture: software layers pile up without any maintained cartographic view, without a unified knowledge. Re-developing is by definition costly since most efforts are repetitions of prior efforts. This approach creates an intrinsic gap between the old and the future new system. How to guarantee that the under-construction new system is the trustworthy image of the old one in terms of data semantics, data processing…? Beyond, what is the degree of interference to the old system when thinking and designing the new one? Ignoring the old one, because it is mainly a wrong source of inspiration, is it the right position? In the other extreme case, copying the old system at the risk of injecting obsolete business procedures in the new information system, is it also a right attitude? In this story, "strict replacement" was eliminated as an alternative.
- Different from "replace", "migrate" amounts to a lightweight or heavyweight transcription. "Lightweight" is the case when one moves from an outdated COBOL (here, COBOL Pacbase whose maintenance by IBM is no longer supported) to a "modern" COBOL. Be careful, "modern" only means actually maintainable through appropriate perennial tools. However, a lightweight transcription is just extending the deadline. In contrast, the transcription can be

qualified as “heavyweight” when the old programs are refactored based on varied concerns (data dispersion or burden, performance, etc.). Heavyweight transcriptions have often the drawback of being offered by the legacy technology provider who proposes her/his “future legacy proprietary technology” as was the case in this story… So, similarly, “migrate” was also eliminated as an alternative.

“Modernize”, the retained solution, is of course opposed to “replace” and “migrate”.

In this story, from COBOL Pacbase to Java EE was an objective orientation despite challenging issues. Java EE had many advantages: vendor-neutrality, durability, worldwide standard, broad support offerings, recognized, proven and widespread technology, cloud-compliance…

Anyway, nobody was naive. Transforming a large-scale COBOL Pacbase information system into a Java EE application bundle was like unassembling a vintage car from the ‘50’s to design a fuel-efficient vehicle keeping the same enjoyable driving sensations being even more important!

4.1 Be in the Cloud, in the close future, no choice?

So, in this story, reconsidering the car was inevitable, but a so-specific driving style prevented the buying of weakly customizable COTS products.

Up-to-date car technologies are appealing, but what is expected is the adaptation of the envisaged car to the company’s driving spirit instead of the contrary. Why not a hybrid engine if sensations remain? More generally, would newest technologies correspond to the driving style, the driving emotion, the driving needs… drivers expect?

In fact, technologies like the Cloud have to be studied but such a survey must not hide the critical porting of the business value from the old to the modernized information system.

In IT, computing infrastructures, including hardware and software, rapidly become brakes instead of facilities. The Cloud seems to offer all the means and inner workings for driving any kind, any shape, any volume of information on infinite highways. This especially includes the absence of constraints like permanently taking care of these infrastructures, upgrading them. It is like all the pleasure of driving a tailored car without fearing the absence of well-paved roads, the absence of numerous service stations, running out of gas or battery and so on.

The X-as-a-service principle of the Cloud (X = infrastructure, platform or software) is orthogonal to the notion of business service already present in any legacy information system. In practice, the Cloud is just a technical alternative that must not hamper the reshaping of the existing business services.

In this respect, clever people wonder: how not to redo the same mistakes? Why Java EE? Why not .NET? Why not Spring? Why not the Cloud? Which cloud? Which will be the legacy technologies of the future. For example, will Java EE be a legacy technology for the Cloud or the dominating technology of the Cloud? Nobody knows because software technology is going faster and faster with unpredictable accidents.

Modernizing towards Java EE or, in a more open-minded way, to the Cloud, beyond a technical debate, is a question of evolvability including standards' compliance, choice of perennial (open) Java EE sub-technologies (*e.g.*, JSP versus JavaFX, JSF, JDBC versus JPA, etc.).

Such a choice through Blu Age® and "models" was fortunately possible. For the present: at least an equivalent information system in terms of functionalities, business execution and possible business growing. For the future: openness, *i.e.*, no more technological adherence because "we already paid a lot to codify our business knowledge in COBOL Pacbase, we do not want to pay twice."

In essence, models are suited to cloud-oriented development due to the flexibility described in Figure 1.

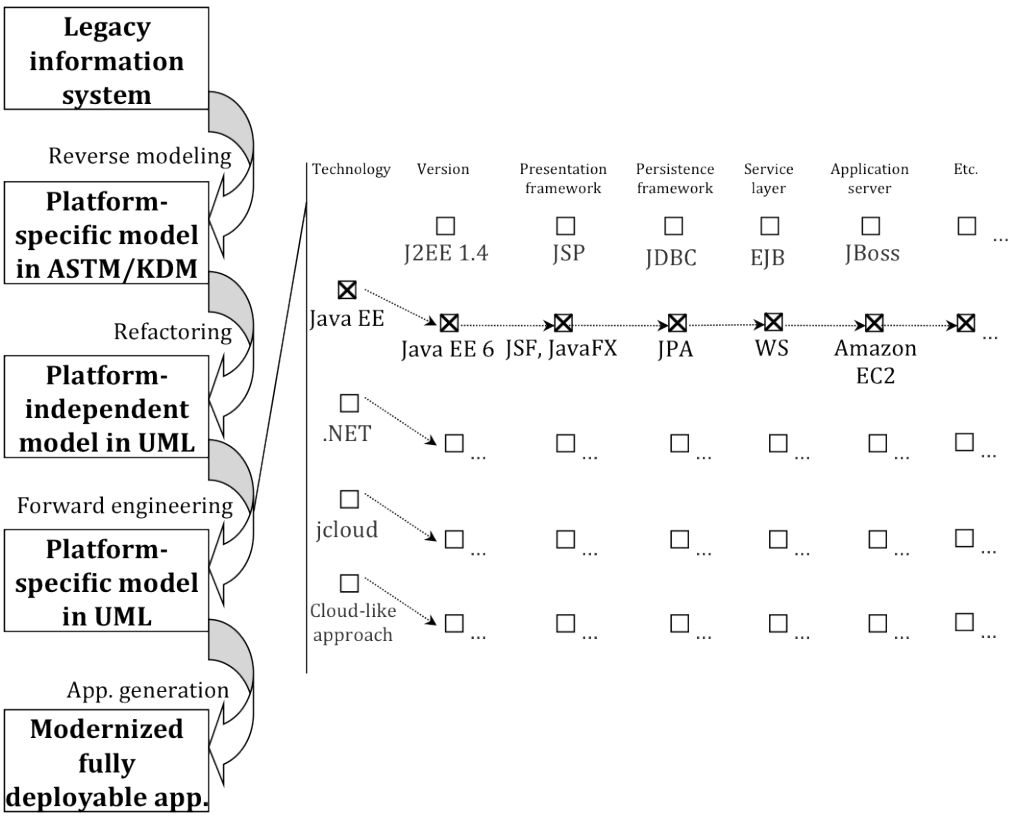

Figure 1. Blu Age® model-driven modernization engineering principle

In Figure 1, the offered model-driven modernization engineering principle shows that UML Platform-Independent Models (a.k.a. PIMs) are constructed from the legacy code. *Refactoring* is the phase when some manual (assisted through model management wizards) intervention on rough models[2] enables the extraction of business data, business rules, business objects (microscopic functions) and services (macroscopic functions). The final results of *Reverse modeling* are these UML PIMs later injected into an application generation workflow, which mainly consists of choosing a technological framework (Java EE, Java EE ver. 6… in Figure 1) including options related to cloud frameworks. Blu Age® backstage is the fabrication of Platform-Description Models (a.k.a. PDMs) technology per

[2] These models are expressed, firstly, in ASTM and, secondly, in KDM. These are two open worldwide standards from the OMG®.

technology, version per version, Web presentation framework per Web presentation framework, etc. This approach, reserved for software architects, gives us the possibility of targeting forthcoming computing platforms. This also strongly guarantees the non dependence of business models with any technological concern when PDMs are woven with PIMs to first produce Platform-Specific Models (a.k.a. PSMs) and next code and configurations files that govern deployment.

4.2 Be agile not agitated!

In this software modernization project, it was not a long, quiet river. Coercive software development processes like the one illustrated in Figure 1, may be confusing to people. Nonetheless, intensive investment in model-driven development must be made compatible with existing practices.

In software development, good practices aim at being codified, to be repeated and reproducible by new players to share a common knowledge, to favor efficiency. Model-driven development is probably a disturbing novelty in the legacy world. Nonetheless, there was a nice opportunity in this project to learn more about software development agility through model-driven development.

There is a remaining confusion about the deep sense behind “agile” in software development. Being agile (a precept!) does not mean being in perpetual movement. Excessive excitement is bad for health, unproductive.

Agility is a state of mind, whether people are young or old. To build and share a common approach, to build a common culture, teams tend to codify their working process in terms of time (phases…), space (documents, software artifacts…) and, above all, acquired and capitalized intelligence on their own business activities, operating methods.

Agility is then a sum of commonsense behaviors: being able to iterate, being able to think “increments”, to anticipate testing, to develop leanly… Opposing existing software development culture and practices to agility is an error. Opposing agility to model-driven development is another serious mistake. None of the other software development technologies is able to create agility as sketched in Figure 1: technology choice rollbacks are possible.

This project demonstrated agility before, during, and after modernization. Domain-oriented agility was the rule in this software modernization project. Effectively, nobody outside the COBOL Pacbase team was capable of reacting to users’ comments, criticisms, requests, continuously assessing, through models, the functional services designed in the new system. In other words, where Blu Age® team brought out “agile legacy modernization” from literature, legacy people brought out some hands-on agility.

Concretely, the lead of the Healthcare Membership Project appreciates Blu Age® team collaborative, roll-up-the-sleeves, attitude that mirrored the local actors’ own culture: “The nice thing about Blu Age® was they embraced [our ongoing delivery experiences] and learned.”

5 Model-driven development in action

Model-driven development has reached the status of mature technology. However, how to remove any fear behind the world "model", literally, an unfinished job? Why? In essence, a model is an abstraction of the system pointing out some system attributes to the detriment of other attributes deliberately absent in the model. Complexity is reduced by representing elements of interest only. Then, models are recognized as valuable supports to think, share, communicate and therefore "design". Unfortunately, such an intellectual (appealing) approach cannot, in most circumstances, be enough. To really design applications in an end-to-end development process, models must be rich enough. All properties of information systems must be made emergent and tangible at a given time in a given model. Separation of concerns is then the most appropriate organization of models in transformation chains (see again Figure 1 for a schematic view of such a chain). In other words, models differ in nature, taking into account the nature of the information they include.

5.1 The human facet of modeling

In the legacy area in general, in this Healthcare Membership case in particular, contrary to popular thinking, people are not intrinsically opposed to the effort of first discovering models and next modeling. Instead, people are open-minded subject to one being able to respond to their "natural" questions. This case reveals model adoption and practice changes:

- Career concerns: people adhered to the idea that model-driven development is an opportunity rather than a constraint; they were able to increase their competencies in model-driven development, *i.e.*, in high tech This opened many other doors like object-orientation, cloud computing… in the sense that training in model-driven development demystified many points: computing remains computing, *i.e.*, models are computing systems. Modeling is a very powerful hammer said Blu Age® CEO, but it does not replace house builders.
- Comfort: doing the same process forever limits risks. Modeling may seem to be different from coding. With Blu Age® executable models, it is not true: the models are the code in a slightly different form, *i.e.*, code is nothing else than a model of bits in computer memory. The key benefit is, with models, the possibility of separating concerns.
- Creativity: being outputs at the very first stage of the software development process, models may be difficult to produce in terms of pure creativity. What was initially considered as a handicap became gratifying. Distinction in roles between business analysts and developers is tenuous in model-driven development. Modeling is thus rapidly viewed as a more noble activity for developers. So, even if modeling is not natural for coders, acquiring knowledge in this area is an expertise increment.
- Loss of control: models as code are subject to long-life maintenance. Investment in modeling is thus equivalent to coding. Model creators are the primary model maintainers. Loss of control is thus a misunderstanding. Code control is not essential because only the models matter.
- The NIH (Not-Invented-Here) syndrome means that developers "reinvent the wheel". Typically, they write their own software instead of reusing software from others. In model-driven development, the final code is generated from models. In this way, one avoids the (re)-creation of already existing algorithms, components, etc. Productivity is the rule! Beyond, this enables to master the quality of the code. Its structure or its complexity may strictly respect high-quality standards like the ISO/IEC 9126 international standard. So,

models might displease developers because they naturally tend to be happy with the NIH. Over time, models are not viewed as a source of creativity deprivation. In this case study, COBOL Pacbase and Blu Age® share code generation as a core development practice. As a result, there was no paradigm shift

To recap, people expected productivity, intellectual satisfaction, pleasure, when using models. Hence, introducing models by this way, is an adequate tactic.

5.2 Model-driven development reality

As claimed before, there was no magic, no miracle. Model-driven development was above all a disciplined activity in this software modernization project. The fabrication of models calls for hard labor, skill. However, model-driven software modernization brings out an important progress: models are fabricated from the legacy code. Manual intervention at *Refactoring* time in Figure 1 is just the incorporation of business knowledge in a fully automated process.

What dominates with Blu Age® is the consistency and the completeness of models to serve as the unique basis for application generation. This is the price to pay for productivity as expected by incomers. In Figure 1, PIMs obey to strict completion rules so that PSMs' elaboration (PIMs + PDMs weaving) and application generation succeed.

This feature is the primary value of Blu Age®. This is not classical in model-driven development. In fact, many modeling tools misrepresent the key foundation principles of model-driven development as highlighted in Figure 1.

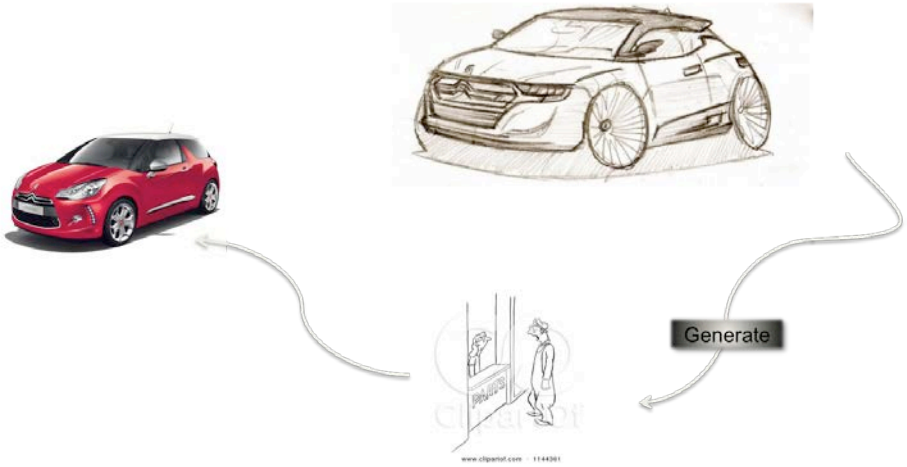

Figure 2. Incomplete application generation in most of the market's modeling tools

As an illustration of this bias, in Figure 2, developers fill in the holes resulting from what has not been generated. In this widespread approach, as the course of time ran on, traceability links vanish, the model becoming, in the very best case, *i.e.*, rarely, a kind of documentation.

In contrast, models may, with additional efforts, be executable. In this case, the model is the code and vice-versa. Blu Age® applies this principle: what you model is what you code (Figure 3).

In Figure 3, contrary to Figure 2, models are very detailed. This is the counterpart of the actual possibility of full application generation. Modeling in such a manner is difficult. This requires open mindedness, professional curiosity, firm conviction and of course intensive training as carried out in this project. Again, do not forget that this Healthcare Membership case is *Reverse modeling*, implying that models are not fabricated from nothing. They come from the legacy code as sketched in Figure 4.

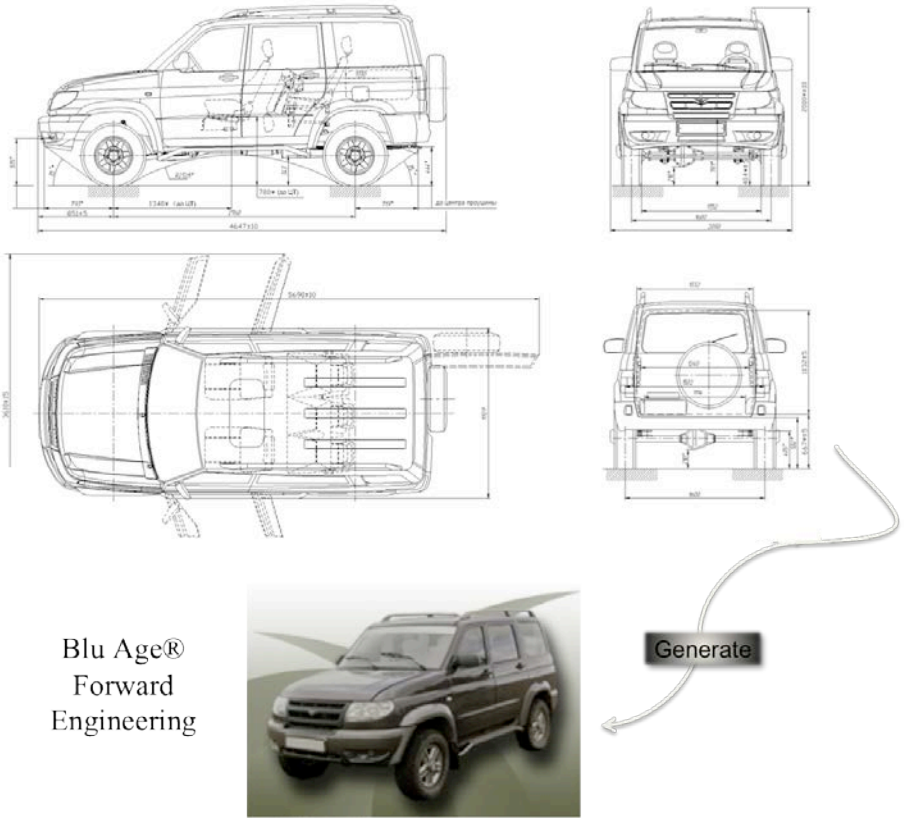

Figure 3. Model-driven development with full application generation

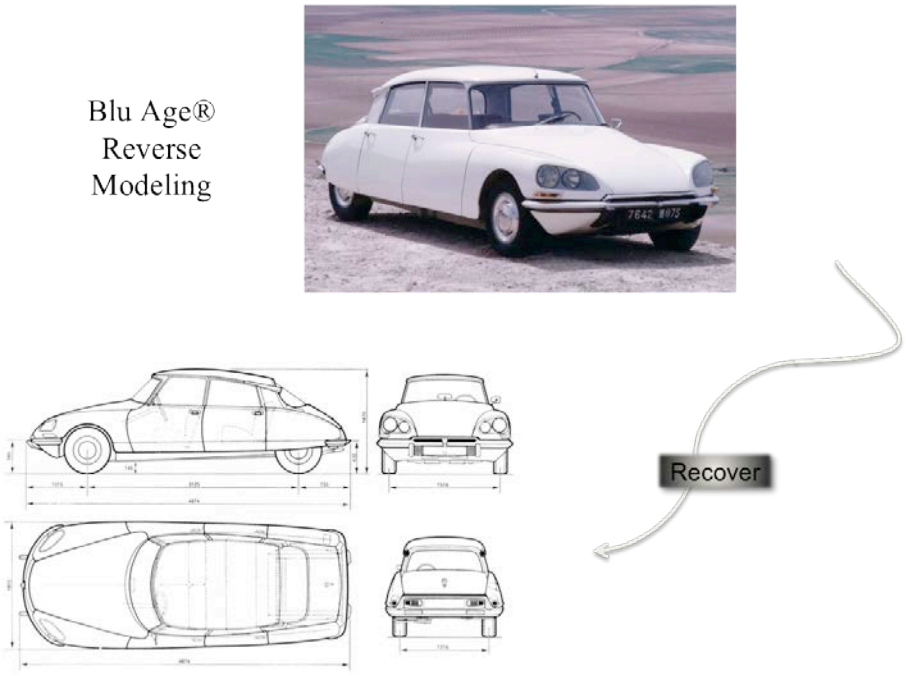

Figure 4. Model-driven legacy information system recovery

What is the recurrent difficulty of model-driven development? It is not really technical. It is rather economical: model-driven development vendors must demonstrate productivity. It is also human. Typically, in the Healthcare Membership case, one must break through human barriers.

6 Conclusion

To sum up, this is not revolution, this is pragmatism and, simply, this is productivity. People investment in any cutting-edge technology similar to model-driven development, must be grounded on simple justifications. Here, from an economical viewpoint, using models is beneficial. Once convinced by this argument, any remaining human barriers can reasonably be destroyed.

In this scope, nothing is new under the sun. Developers for a long time have been model followers (not on Twitter!) on a daily basis when they construct programs: a suite of statements in a rigid language (COBOL, C…) only understandable by a computer. What developers do not, is provide “0” and “1” suites to computers to express their orders. So, **they really were, they are, modelers**. Contrary to developers, software architects and business analysts have taken more an overview in their everyday work. They have to understand and control why and how “0” and “1” are circulating in software architectures and business processes. What is new (and a growing source of innovation) is that models may create a convergence for these three key computer job categories. All three may now share their intellectual production through “models”.

In this story, models were in effect a cultural rupture, a (rapidly vanishing) psychological resistance, and, at the end, the most powerful software toolbox people encountered in their software life. Forever, professional inertia may tend to regard that models are a source of problems, the ultimate paradox: before any practice! Real-life business as it happened in this Healthcare Membership project demonstrated the contrary. The jump to models was the motivation to embrace high tech,

not “high tech for high tech”; it was high tech for leveraging legacy knowledge in order to smoothly transform it into renewed software assets prepared for the future.

7 Blu Age® solution advantages

As a summary, Blu Age® is a sum of self-contained functional components (*Reverse Modeling*, *Forward Engineering*, *Database Modernization* and *Factory*) to carry out an end-to-end software modernization project as described in this paper for a large-scale Healthcare Membership case study. The key benefits of this solution are:

- **There is no adherence between Blu Age® and the modernized application produced by Blu Age®**. Models representing business logic (data as entities, business objects, *i.e.*, small and modular functionalities, services as well calling business objects and embodying larger functionalities) are expressed in normalized formalisms (*i.e.*, ASTM, KDM and UML within Eclipse). Namely, UML models ready for application generation represent, in a totally agnostic way, the modernized application. The database and code base resulting from generation do not require any specific runtime sold by BLU AGE CORPORATION.
- **Blu Age® hides the building of models**. In model-driven development, drawing models from scratch is not easy and straightforward. Blu Age® Reverse Modeling assists and guides users when interpreting the legacy code. For example, this component offers a learning mechanism that allows the identification and demarcation of code patterns. Patterns are often code templates. Then, the engine of Blu Age® Reverse Modeling may automatically process legacy code (well-delimited) blocks into components able to be integrated into models.
- **Blu Age® manages target execution platform in neutral way**. Execution platforms are PDMs woven with PIMs to obtain PSMs. This approach lets the possibility of generating toward any platform, including platforms of the future like cloud platforms to anticipate upcoming modernization. The Blu Age® Forward Engineering component is thus fully independent of PDMs while it always capable of transforming a PIM into a PSM linked to a given platform.
- **Blu Age® may be customized to marry homemade software development culture and practices**. Blu Age® is above all a set of engines (model transformation programs) and intelligent wizards. The Blu Age® Factory component is in charge of defining tailored modernization workflows, including agility concerns: iterations, increments, early validation and evaluation, etc.
- **Blu Age® is scalable in particular for dealing with bulk models**, a common situation when models represent legacy software artifacts. Blu Age®, differently from other (open source or not) solutions, has been experimented, developed and tested in relation to significant and representative case studies.

8 List of acronyms

- ASTM: Abstract Syntax Tree Metamodel
- EMF: Eclipse Modeling Framework

- KDM: Knowledge Discovery Metamodel
- OMG: Object Management Group
- PDM: Platform-Description Model
- PIM: Platform-Independent Model
- PSM: Platform-Specific Model
- UML: Unified Modeling Language

9 Bibliography

S. Selip, *Healthcare Membership System Modernization - COBOL Pacbase to JEE*