Recent advances in IT architecture : some guidelines

Recent Advances in IT Architecture – some Guidelines

In the autumn of 1968 and 1969, NATO hosted a conference devoted to the subject of software engineering in the light of the trouble encountered by the computer industry in producing large and complex software systems. In the meantime we have experienced many advances of technological and sociological nature that impact software engineering. Despite these attempts at finding the silver-bullet of software manufacturing, writing enterprise applications has not become simpler. In fact, software engineers must close the gap between software technology aspects and complex and ever-changing business requirements and between integrating legacy systems with brand-new off-the-shelf components.

The programme "Software Technologies for Advanced Internet Services" explores new opportunities for Swisscom arising from current software technology trends and assesses the impact these trends may have on the efficiency of the service creation process as well as on the quality of the services created using those technologies. The focus of the JEDI project is to explore trends in software engineering in the domain of software architecture.
With its Innovation Programmes, Swisscom Innovations follows the objective of recognising early on the impact of technological developments, finding new business opportunities, promoting technical synergies, and developing concrete innovation proposals. Further, the expertise built up enables active engineering support of business innovation projects.

FRANZ ACHERMANN AND
BRUNO T. MESSMER

More than thirty years after the famous NATO conference in which the term "software crisis" was coined, we are still far away from being able to build large, mission-critical, complex software systems on time and on budget in an "engineered manner". More often than we like, our software projects do not follow a well-engineered path, but we stumble from the solution of one problem to the next. When we succeed, this is more often due to the fact that there is a genius programmer in the team than due to our controlled software engineering process.
The term "IT Architecture" is increasingly recognised as one of the most important aspects to make software development more of an engineering task. The architecture commonly includes code artefacts, the blueprints, the software process steering the development and evolution of the project. However, the real benefit of IT architecture only unfolds when its ideas and concepts can be successfully communicated to the development team.

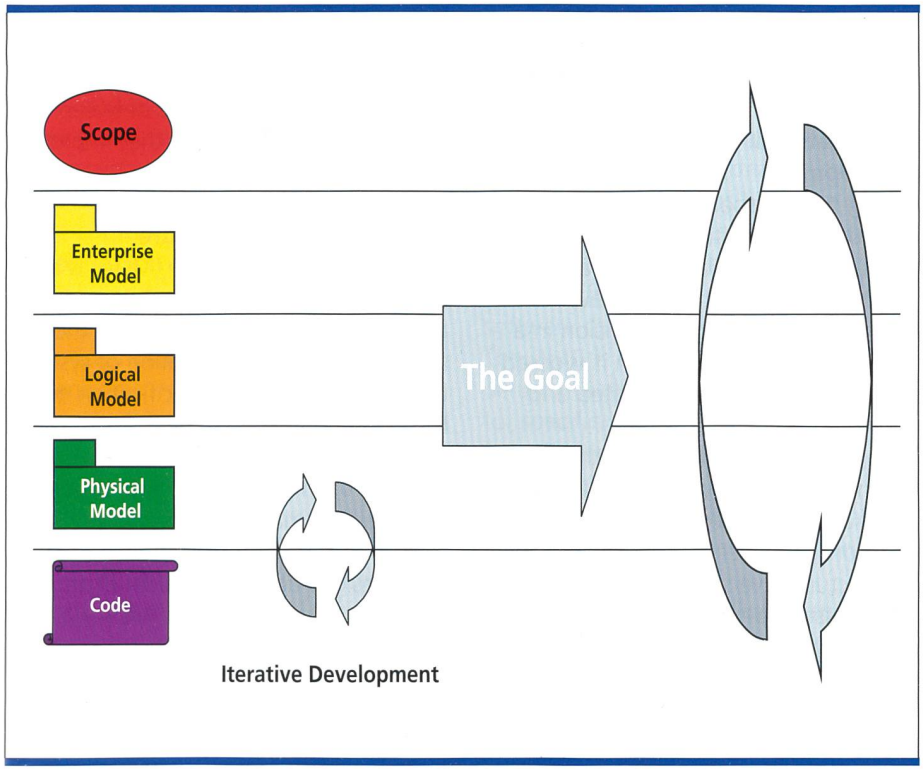

Fig. 1. It is one of the goals of research in IT Architecture and agile methods to allow iterative development over all layers. Currently short feedback cycles are mainly supported for the (late) development and testing phases.

Hence, the quality of an architecture not only depends on its actual content, but also on its quality to be communicated. In this article we present some emerging engineering approaches and techniques and explain how these techniques help to communicate the architecture of complex IT systems.

All mature schools of art, engineering and science have their own special languages which have evolved over the years. These languages help experts to express themselves more accurately. In the context of software systems, these terms are organised more and more using the vocabulary of architecture.

The word "architecture" has become a central word in the software business. It is a term that lots of people define with little agreement. When people describe their software architecture they select the important parts of their systems, how these parts fit together, and the key decisions they made in designing their systems. Architecture is the shared understanding of a system's design by the expert developers of a project. This understanding is typically given as an architectural model in the form of the major components and how they interact.
More important than a good architecture per se is – we argue – the fact that a system has an adequate architecture that can be communicated.
Communicating the architecture is the key enabler for:
– Software evolution
– Iterative development and thus more agile development
Being able to communicate, the architecture is central for a system to survive. The first law of Lehman, the law of continuing change, says that any software system used in the real world must change or else becomes less and less useful in that environment [1]. The architecture of a system is not something static that once was genuinely designed and then stays there for the rest of time. It is a process of constant updating and keeping in sync with the business models and with the technologies used.
Often, the original designers are not available when a change is designed and implemented. Thus people without the origi-

nal insight apply changes. If they do not fully understand the architecture, changes and patches are often applied in the wrong spots in the program and lead to "architectural drift". The overall structure of the systems gets hidden more and more and becomes complex.
Communicating and making the architecture understandable also helps to track business requirements. As business requirements change, it is important that we can actually trace down how a business rule is implemented in the final system.

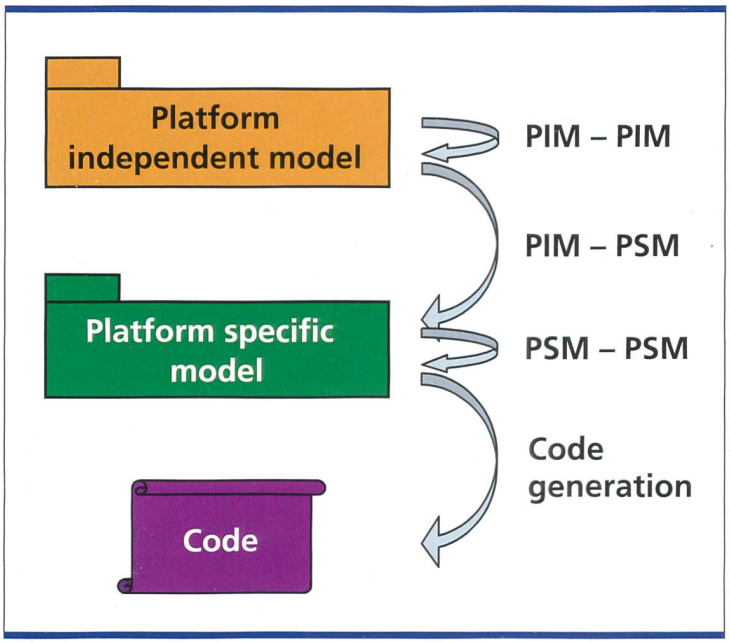

Fig. 2. Transformations on models in MDA.

Levels of Abstractness

Abstraction is the process of leaving out irrelevant details. In order to communicate the architecture of a system, models are needed at every level of abstractness. At every level different architectural issues must be resolved and yield different models using a different terminology.
The Zachman framework for enterprise architecture [2] distinguishes the following levels (with decreasing level of abstractness):

- *Scope* is the industry view, concerned with the things that define the nature and purpose of the business.
- *Enterprise* defines the nature of the business, including its structure and organisation using business terms and functions perceived by the business experts.
- *System* defines the business in more rigorous information terms.
- *Technology* describes how technology, such as XML or relational databases may be used to address the information processing needs identified in the system model.

An enterprise architecture is considered to be convergent when the different abstraction layers are in sync. For instance, the business objects identified in the enterprise level are typically represented by system entities on the system level and correspond exactly to a number of tables in a relational database scheme.
In the light of the different abstraction layers, communicating the architecture becomes a new important aspect: it is not only crucial to communicate and evolve an architectural model from one version to the next, but is equally important to communicate the architectural model when going from one layer to the next. Communicating horizontally, i.e. over time from one version to the next is simpler, because we expect people with a similar background and similar training in their vocabulary.
In contrast, vertical communication is much harder because the people involved often come from a different background and use different words for similar things. When moving and communicating the architecture between abstraction layers, not only technical aspects come into play. Every abstraction layer has its own experts, thus at the borderline of layers, people with a different background must co-operate and inform the peer party of "their" architecture.

From Waterfall to Iterative Development

During the life span of a project from inception to transition a system cross-cuts the different abstraction layers.
The project starts at the top level as an explicitly expressed goal or vision in a business context. This vision is worked out more and more, elaborated and implemented.
Every project is challenged by the fact that not all problems that may appear at lower abstraction levels can be foreseen. As a consequence, in order to keep the architectural layers in sync, the modelling team must step back, adapt the requirements and resolve the problems. The software engineering community uses the term "iterative development" for this process. For the implementation and testing phase, iterative development is well accepted and a good tool support exists to make the development process more agile.
A typical symptom of iterative development, however, appears when feedback is not properly integrated into the development process and the corresponding documentation and design documents created are not in sync with what is actually developed and deployed.
One of the goals of research in IT architecture is to understand the principles and find out methods that enable iterative development over all abstraction layers (fig. 1).
In the following, we will describe some recent trends and compare how they help to communicate the models between the abstraction layers and how they improve agility on the whole process.

Model Driven Architecture

Model Driven Architecture (MDA) is an approach proposed by the Object Management Group (OMG) based on the separation of the specification of system functionality from the specification of the implementation of that functionality on a specific platform.
The approach promises a large productivity gain by making models the primary software artefact, thus protecting software designs against frequent changes in realisation technologies. MDA builds on the insight that architectural models can be described using UML and that these formal descriptions can be transformed. Such a transformation refines certain aspects of the architecture.
A transformation step thus adds or implements additional information to a model.

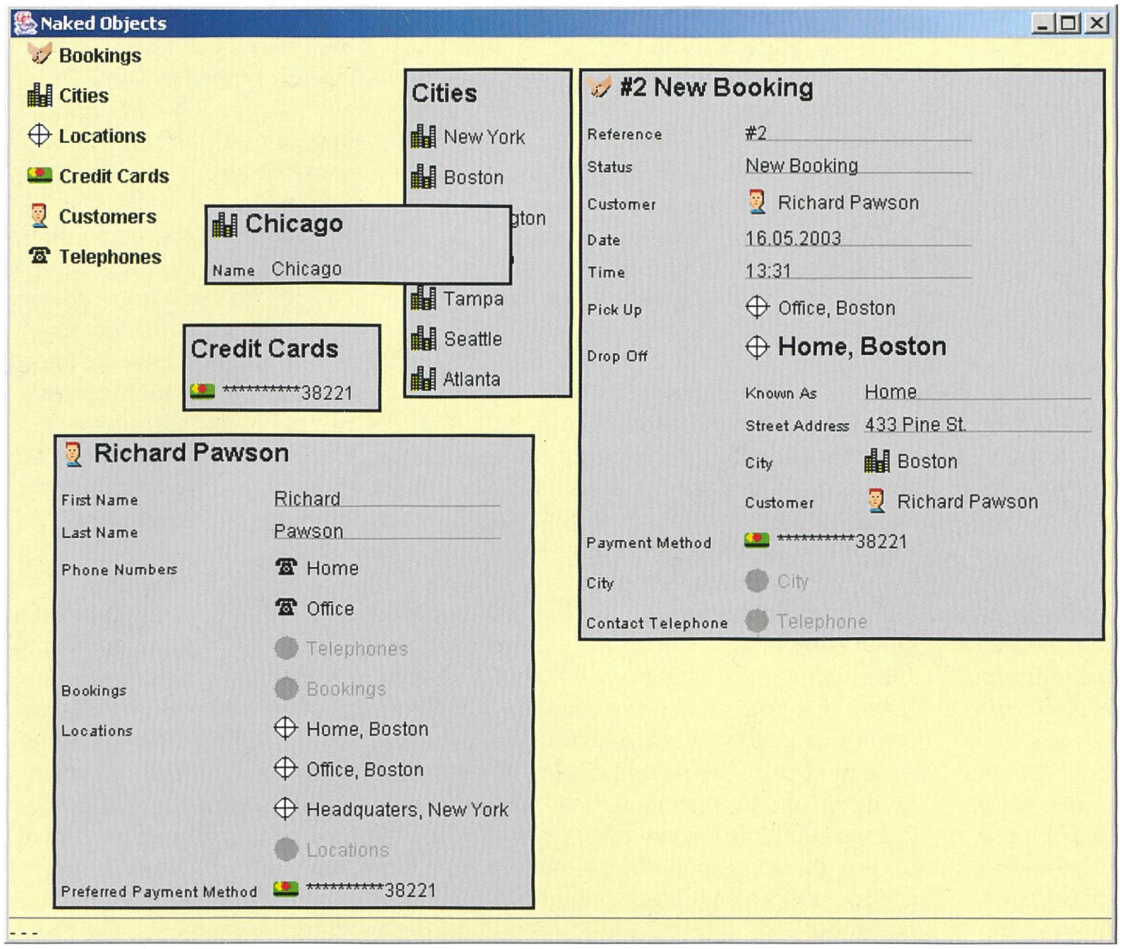

Fig. 3. A snapshot of a typical Naked Objects application. Displayed are several instances of business objects with their attributes. The "New Booking" object on the right was created by invoking the "new Booking" method on the business object "Richard Pawson" on the lower-left corner. Note that, for instance, the preferred payment method of the customer is set as default payment method for the new booking.

MDA defines three types of model: PIM, PSM, and running code.

- The platform-independent model (PIM) is a formal specification of the structure and function of a system without any technical details. The PIM model corresponds to the system model of the Zachman framework.
- The platform-specific model (PSM) is a specification of the structure and function using technical, i.e. platform specific details. The PSM model is part of the technology layer in the Zachman framework.
- The running code is the fully functional code, expressed in a programming language.

By introducing PIM, MDA makes it easier to validate the correctness of the model as it is uncluttered by platform-specific semantics. For example, PSMs have to use the platform concepts of exception mechanisms, parameter types (including platform-specific rules about objects references, value types, semantics of call by value, etc.), and component model constructs; the PIM does not need these distinctions and can instead use a simpler, more uniform model.

MDA proposes several ways to transform a platform-independent model into a platform-specific model. These proposals range from the manual transformation done by a platform expert, to the fully automatic conversion. An MDA tool typically assists this transformation by the generation of skeletons based on the user input. Whether complete automatic generation of platform specific model can be achieved is heavily debated in the software community.

The OMG is defining platform independent meta-models (using UML profiles) for specific domains, including the telecom, finance, and e-business domain. For instance, modatel is a current IST project on MDA for Telecommunications System Development and Operation [3].

How MDA improves Communication

Since MDA is about standardising the models that are exchanged, it improves communication between different tools. The standards defined by MDA will enable business process modelling tools to communicate with designing tools.

From a more technical aspect, MDA speeds up the development process as a number of tools are available that automate the transformation more and more.

Open Issues

The intent of MDA is that models are only transformed from the abstract to the more concrete levels. However, the iterative development process also requires that these transformations work in both directions, i.e. not only from the more abstract to the concrete models, but also in the other direction. However, this is still a vision and most tools support only refinements and not abstractions.

Naked Objects

Naked Objects was presented at the most recent OOPSLA Conference in Seattle. OOPSLA is one of the largest annual conferences on software engineering and object-oriented programming. Larry Constantine writes [4] that "the most significant event was not in the regular program but in the demo track, where [...] Richard Pawson and Robert Mathews [...] peddled their Naked Object approach to the problem of user interface design. Their solution for usability? Eliminate the user interface design altogether".

The core idea of a Naked Objects system is that a business application appears to the user similar to a drawing application. Instead of colours, drawing objects and pens, however, the user has access to the business objects such as customers, contracts and services. These business objects are presented visually and can be manipulated directly ("naked") by the user, instead of being hidden behind menus, forms, process-scripts and an army of dialogue boxes.

Instead of designing, building and testing the user interface for a software system, all the developer needs to do is to create software objects that correspond to and fully model the business objects that make up the application domain.

The current version of a first implementation of the Naked Objects approach is a Java framework that uses Java's reflection facilities to auto-generate the user interface from the business objects. The basic idea is that the user invokes methods on the business objects using pop-up menus and that he can modify the fields of business objects. The framework takes care of providing functionality to create new objects, to make them persistent and to provide finders to retrieve them again, and to provide access to the objects using thin-client architecture. Figure 3 contains a snapshot of a naked objects application using the most recent version of the framework.

Using the framework a developer can, for instance, implement a fully operational toy bank application consisting of users and accounts. All he needs to code are two Java classes. Contrast this with the EJB or Corba examples, where the business logic is the same, but in addition to the core business objects, the developer also needs to implement the user interface, the stubs and proxies for the middle-tier etc.

How Naked Objects improves Communication

One of the strongest points of the approach is its reliance on and intensive involvement of the user during the collaborative development phase. The approach helps the user to understand object-oriented concepts like objects, classes and associations because these concepts can be seen and manipulated in real time. Once the user understands that certain business elements like customers are mapped to objects, they begin to formulate precise requirements on how to improve the system, as the following quote illustrates:

"After only a few hours of involvement in a Naked Objects project, the users start to express their ideas and requests for new functionality in object terms: 'Would it be possible to have a new action on the Promotion object to visualise the leaflet?', 'We need another sub-class of Store to represent our Petrol Filling Stations' [...]" [5].

Clearly, the tight contact of the user with the actual business objects of the program reduces the intellectual distance between the system designers and programmers and the users and, hence, will allow more precise communication on the system's purpose, requirements and functions between these involved parties.

Open Issues

The greatest usability problem with Naked Objects is the one-size-fits-all premise on which the approach rests. Instead of tailoring the presentation of information and the operation to the user interface to fit the unique aspects of the context, the application and the user needs, one solution is presumed to fit all problems.

As the user-interface is auto-generated, it always looks the same. All interaction with the business objects follows the same pattern. As an example, consider how printing is done in a traditional application. If a user wants to print, he invokes the print menu. A dialog box appears where the user can choose among the available printers and what to print. Clicking on the "ok" button starts the printing process. In contrast, in a Naked Object system, the user first selects the objects to print, (e.g. a bill or a chapter of some document) and then drags this object to one of the available printers. Such an interaction style contradicts and replaces much of the user-interface design people are familiar with, and it is yet unclear whether this approach is only useful for prototyping or whether it might also be applied to productive applications.

Conclusions

We have looked at the two recently introduced new techniques from the field of software architecture, Model Driven Architecture and Naked Objects. We classified and estimated their relevance in how these approaches help to communicate intentions and design of a system between the different abstraction layers. The two techniques differ in their origin as well as in the industry support:

- MDA is heavily pushed by OMG and tool vendors as the new way to build software systems. In the near future, we will certainly hear a lot about MDA, since tool vendors will classify their development tools as "MDA compliant".
- Naked Objects is a fairly new topic. The approach currently has a few followers. The future will show if and how some of these ideas and principles get into the mainstream development process. First experiments with this approach have shown that it may help to better involve the user and his needs in the prototyping and the requirements analysis phase.

Recently, the common understanding that running code is the main deliverable of a software project is slowly shifting toward the notion that the architecture model should become one of the prime deliverables of a software project. What started out with the inception of simple design patterns as higher-level building blocks is becoming a general movement of the IT industry towards design and implementation approaches which reduce the gaps between the models and the running code and therefore increase the communicability of software systems. 10

Abbreviations

MDA	Model Driven Architecture
PIM	Platform Independent Model
PSM	Platform Specific Model
Architectural Drift	The effect that happens to the software architecture over time when applied changes more and more obscure the initial design
OMG	Object Management Group
UML	Unified Modelling Language

References

[1] M. Lehman: "Programs, life cycles and the laws of software evolution", Proc. IEEE, 15 (3), 1980.
[2] http://www.zifa.com
[3] http://www.modatel.org
[4] http://www.foruse.com/articles/nakedobjects.pdf
[5] R. Pawson and R. Matthews: "Naked Objects", John Wiley & Sons, Ltd. 2002.

Pointers

The official MDA webpage:
www.omg.org/mda/
Naked Objects:
www.nakedobjects.org

Franz Achermann *works as a Software Engineer at Swisscom Innovations. He has been involved in J2EE application projects and database applications as programmer and consultant. He holds a PhD in computer science for his work on software composition.*

Bruno T. Messmer *is Programme Manager at Swisscom Innovations, responsible for the Innovation Programme "Software Technologies for Advanced Internet Services". Before joining Swisscom in 1996, he received a doctoral degree from the University of Berne for his work in the area of pattern recognition and graph matching. He has published over twenty articles on the subject of efficient graph matching algorithms and software frameworks. Furthermore, he has an ongoing interest in object-oriented technologies, knowledge management and the semantic web and, in general, the application of Artificial Intelligence techniques to the telecommunication domain.*

Zusammenfassung

Seit dreissig Jahren spricht die IT-Industrie vom Phänomen der «Software Crisis», die sich in erster Linie darin manifestiert, dass bis heute lediglich 5% aller Software-Projekte innerhalb des Zeit- und Finanzbudgets erfolgreich beendet werden konnten und über 50% aller Projekte abgebrochen werden mussten. Die Suche nach dem Stein des Weisen der Software-Entwicklung produziert deshalb seit Jahren verschiedene Vorschläge zur Verbesserung der Situation, jedoch bisher ohne durchschlagenden Erfolg. Ein vielversprechender Ansatz könnte in der neuerlichen Belebung der Begriffe «Architektur» und «Modell» liegen. Dabei beinhaltet «Architektur» nicht nur den eigentlichen Code, sondern auch Entwicklungsprozesse, Entwurfsmuster und Vorgehensweisen. Ebenso wichtig wie die Architektur eines Systems richtig hinzukriegen ist es, die Architektur zu kommunizieren. Kommunikation läuft dabei in zwei Dimensionen ab.

– Erstens wird im Wesentlichen die Architektur eines zu erstellenden Systems zwischen den verschiedenen Interessensgruppen kommuniziert. Es wird also die Vision eines Systems in den Geschäftseinheiten artikuliert, dann an die Systemarchitekten weitergegeben und verfeinert und schliesslich von den technischen Experten auf einer Plattform umgesetzt.
– Zweitens wird die Architektur auch im Lauf der Zeit weitergegeben und kommuniziert, wenn eine neue Funktionalität hinzukommt oder neue Teammitglieder eingearbeitet werden müssen.

In diesem Artikel werden zwei komplementäre Ansätze vorgestellt, die in diesem Zusammenhang von Bedeutung sind, nämlich der «Model Driven Architecture, MDA»-Ansatz von der Object Management Group und der «Naked Objects»-Ansatz. In MDA wird die Vision vorangetrieben, Software-Artefakte als Modelle zu sehen und diese ineinander zu transformieren. MDA-fähige Werkzeuge erlauben es beispielsweise, aus logischen Systemmodellen Code(-fragmente) für eine bestimmte Plattform zu erzeugen. Anstelle des Code-Generierungsansatzes von MDA propagiert der Naked-Objects-Ansatz, direkt die Geschäftsobjekte zu manipulieren. Naked Objects generiert zu einer Anzahl Geschäftsobjekte automatisch eine Applikation. Damit erlaubt Naked Objects, sehr rasch aus der logischen Sicht eine lauffähige Applikation oder zumindest einen Prototyp zu generieren. Beide Ansätze verbessern so die Möglichkeiten, die Architektur eines Systems zu kommunizieren.