Software Improvement Through Model Transformation (Semantic Technologies)

Computer Programs get older and finally they retire. For a software system there are many reasons, why it should/can not be in use anymore. Sometimes it is the technology, which develops faster and overtakes the capabilities of an IT system, such that programs cannot satisfy its organization's needs to remain competitive. For example, if a program cannot be deployed to web, it may be considered to be unsuitable for the usage, since most of the IT services must be available in the Internet medium as well as in internal nets of the companies and at the same time on devices like PDA's too. In some other cases the missing quality of IT maintenance because of retired employees and steady grooving systems leads to programs which are of poor quality. Organization's IT becomes a so called legacy system.

Sometimes it is the software architecture or the paradigm, which plays an important role in the decisions of IT managements. The basically procedural programming languages like COBOL, PL1 and FORTRAN are now older programming languages, which are being replaced by more modern ones day by day. The object oriented software paradigm is state-of-the-art because this type of programming is considered as more efficient and user (developer) friendly by experts. The steady rising complexity of the technical infrastructures force some other companies to change over to Code Generators like CA Gen (market leading product of Computer Associates).

The question which needs to be answered is, when a programming language becomes an old one. As stated above, various parameters influence this decision. One important decision criterion is the continuing education and availability of related IT staff and the ability of the programming language to support state-of-the-art technologies. The fact that its modelling interface is not contemporary is mostly a reason not to use a programming language, since the modelling interface has a direct impact to its usability and efficiency. Finally, the agility of the software provider against changes and whether he would be able to deliver needed changes on time (time-to-market) or not is a crucial fact.

Software Modernization

The most common solution of the stated problems is a migration of the IT system into a newer technology.

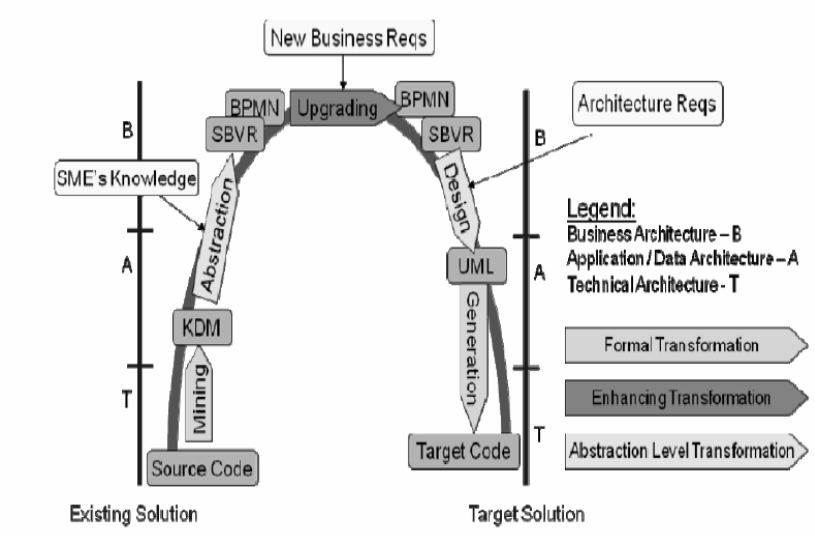
The migration of an IT infrastructure to another one is not a trivial task. The technology which relies behind these efforts is based on model transformation. A model is an abstract representation of a system, model in science is a physical, mathematical, or logical representation of a system of entities, phenomena, or processes. Basically a model is a simplified abstract view of the complex reality.The rules and constraints to which the elements of a model must obey make up the metamodel of that model. A programming language or the whole IT system can be described in terms of their metamodels. The model transformation needs beyond the metamodels(M2) a common meta-metamodel(M3) of the participating models. The same meta-metamodel makes a transformation possible. If the source and target models wouldn't be of the same kind (of the same M3), the transformation would not be easy. For example in ECLIPSE the compatibility of the metamodels to ECORE makes the wok much easier. The metamodels of the systems which participate at a transformation process are not always available. There are some techniques with which the metamodels can be captured. The Backus-Naur form of IT systems are widely used to create

parsers. From the grammer of a language its metamodel can be generated. Another technique is the usage of a metamodelling toolkit with which the behaviours of systems can be traced.

In OMG's terminology systems modernization relies on an architecture driven transformation. Basically there are three types of architectures. These are technical architecture, application data architecture and business architecture. Technical architecture describes technical infrastructure of the system. The application and data architecture gives a blue print for data persistence and the programs. Business architecture consists of the process flows and describes how the processes are build. The modernization process is made up of some transformations which must be formal. Formal transformation means that the rules are mathematically well defined. Otherwise the generation of the target language is not possible.

The first step is the parsing of the source language. The second step is the abstraction (generalization), which is followed by the third step enrichment. The method is the analysis of the source language and the construction of an instance of a so called Knowledge Discovery Metamodel (KDM). The KDM stores every necessary information about the system. If this once stored, further transformations lead step by step towards the target system. Here is the knowledge of a systems specialist inevitable, because the abstracting transformation and the following enhancing transformation to build the target system require his know-how.

OMG calls this transformation a horse shoe transformation since the figure in which this transformation is depicted looks like a horse shoe.



OMG's Horse Shoe Model

A framework for model transformation ProgGen (Prog[ram]Gen[erator])
==========

The transformation of a system from a source to a target language is some sort of rejuvenation. It is a decomposition of all 'cells' of a system and the rebuild under a new context. It is a semantic transformation. The system must be aware of the meanings of the concepts which are part of the system to be transformed. The language in which the source system is written must be understood by the transformation framework. Otherwise is a correct transformation not possible. The transformation can also be done into same system like the

source. Some restructuring of the same system can be targeted. For example an adaptation for SOA (service oriented architecture) can be achieved. This requires the separation of the business logic from the other program parts like presentation. In some older systems the business logic and the presentation logic are not separated from each other. That's the reason why all applications which drive the daily business cannot automatically be used in some other context for PDA's or some other modern state-of-the-art technologies and interfaces.

There is not just one way of system modernization. System modernization is always decided by each organization IT architects, depending on the strategy of this organization. This also is the reason why in the horse shoe model in the abstraction part the knowledge of the enterprise and in the design part the knowledge of a systems architect needed. The formal transformation of the source system is rather a straight-forward activity. The abstraction of it is a little bit harder since the parsed source system must be brought into a neutral abstract shape which preserves the semantic (meaning) of the programs but not necessarily the way they were realised in the source system. For example the system knows and stores that some file system captured information but not the exact commands like get, put, etc.. how the file accesses were performed in the source system.

Another important aspect is the change of paradigm among the source system and the target system. If the source is a procedural language like PL/1 or COBOL and the target for example JAVA the importance of the abstraction (generalization) is even more. In this case the abstraction must also include the transformation of the procedural paradigm to the object oriented paradigm.

It means that the classes must be built, the methods and attributes must be determined. Finally the transformation of the logic must be accomplished. The support by the underlying framework at this stage is very important.

The framework must support decomposition and rebuild and transformation. The Decomposition strategy can again be defined by the conversion team. For example in case of a COBOL or PL/1 program the sections or procedures can be considered as the smallest program components. The framework should support this decomposition and then the composition of the components to greater units must be possible. In OO transformation are they packages.

The RHS of the transformation is the creation of the target system. Once the target system's metamodel instance is populated tool adaptors persist the result. It means that they create the target system.

Transformation Technique
==========
The transformation occurs among the metamodel instance on the LHS (source) and the metamodel instance on the RHS (target).
Matching (Bridging) happens in different ways. Here is the applied strategy the crucial issue. If the migration will be a 1:1 migration, which means that the logic from the existing system will be transferred to the new one in the same scale without any optimization of information flow and programs and if even the paradigm is the same, the work is straight forward the transformation happens translating the semantic of the source almost line by line to target. The bridging is not that easy, if the transformation strategy requires an upgrade of code quality. In this case the creation of target instance requires much work. The creation requires pattern transformations, variable changes and eliminations, code restructuring and

enhancements and understanding of the source. It means ‘hot dog’ must be translated as ‘sausage sandwich’ not as ‘a dog which is heated up’.

The degree of difficulty rises even more, if there is also a change of paradigm. The system must in this case learn and use the knowledge of the architect to create the transformation rules. The used technique for this stage requires the usage of ontologies of related programming languages. The ontology should be built by the systems architect if it doesn’t already exist and is being consulted by the system each time if the transformation rules are created. Such kind of technology is called ‘semantic technologies’.

ARIKAN Productivity Group finished in december 2009 extensive research and development in related areas in the ModelCVS *) project starting 2004. For further information in model transformation and software modernization write to mustafa.arikan@arikan.at

Mustafa Arikan , Vienna 24.2.2009

(*) This work has been awarded and partly funded by the Austrian Federal Ministry of Transport, Innovation and Technology (BMVIT) under Grant FIT-IT-810806.