

What is needed for MDD Success – Experienced Technological Shortcomings

Bjørn Nordmoen¹, Tor Neple²

¹ WesternGeco, Solbråveien 23, N-1383 Asker, Norway

² SINTEF ICT, Forskningsveien 1, N-0314 Oslo, Norway
¹ nordmoen@oslo.westerngeco.slb.com, ² Tor.Neple@sintef.no

Abstract: Over the last decade Model Driven Development (MDD) has attracted a lot of attention within the Software Development community. Models are typically used to create (through transformations) other artefacts such as models or source code. There currently exists a multitude of different tools and technologies to perform the different parts of MDD, however few, if any, address the whole spectre in an integrated fashion. This paper presents key challenges to the industrialisation of MDD through describing areas where the authors believe there is potential for improvements of the technologies based on experiences of using MDD and traditional software development.

Keywords: MDD, Tools, Experiences

1. Introduction

Over the last decade Model Driven Development (MDD) has attracted a lot of attention within the Software Development community. The large scale industrialization of MDD however has not yet happened. Still models are mostly used for analysis, design and documentation.

There currently exists a multitude of different tools and technologies to perform the different parts of MDD, however few, if any, address the whole spectre in an integrated fashion. In this paper we present some key technological impediments to MDD adoption based on the current state of the industry.

This paper presents work in progress, and reports the findings of WesternGeco as an end user partner in the MODELPLEX project. The issues presented in this paper are those that the authors mean are the most important ones for MDD to become successful. These issues have been brought as input to technology development in the MODELPLEX project

1.1. About WesternGeco

WesternGeco works in the area of Oil and Gas Exploration, offering advanced seismic services. The aim of seismic surveying is data acquisition to produce images of geological features and their structure below the surface of the earth. The company

2 Bjørn Nordmoen 1, Tor Neple 2

goal is to provide a full seismic service package to the oil companies, ranging from acquisition via processing to interpretation.

In order to increase differentiation, WesternGeco's strategy is to develop proprietary seismic-related software and hardware products to support the seismic service business. The technology covers the range from high-speed embedded sensor- and data-collection networks to advanced instrument control and data processing on super-computers and large clusters. The main features of this type of system are real-time, large data volumes and CPU intensiveness. These systems are also highly technology bound and will therefore face the prospect of being continuously upgraded for the next 5 to 10 years.

The WesternGeco systems comprise a multitude of platforms and development is done using various technologies and languages such as CORBA, RMI, Java, C++ etc.

For the last 8-10 years WesternGeco have taken part in a number of projects related, in some way, to Model Driven Development (MDD).

1.2. Structure of this paper

This paper is structured as follows: Chapter 2 presents a detailed description of the technology areas where we experience shortcomings. Chapter 3 includes a discussion on the importance of Open Source while chapters 4 and 5 present future work and conclusions.

2. Challenges for industrialization of MDD

Having experimented and worked with MDD and related technologies for quite a number of years we have seen a clear evolution of the available tools and technologies. There are however still areas of technology that have shortcomings that we believe are hindering the potential success of the MDD paradigm. We have grouped the technologies into two areas; essential and value added. The Essential area describes what we believe are features and technologies that absolutely need to be present, while value added are things that would be nice to have.

2.1. Essential Technologies

2.1.1. Industrial strength Integrated Development Environment

Software development processes include a number of different tasks that need to be supported by functionality from tooling. This is also the case, probably even more so, for MDD. It is essential that these tools are of industrial quality and comprise a rich set of features.

Ideally a low cost (open source) modelling tool suite should be available comprising the following functionality:

- A requirements management tool

- A UML modelling tool and/or DSL modelling tool
- A Testing tool /framework
- Model transformation tools (Model to Model and Model to Text)
- Traceability capability (from requirements to model-element to code to tests)
- A transparent model storage facility providing functionality as found in Source Control Management systems such as Subversion.

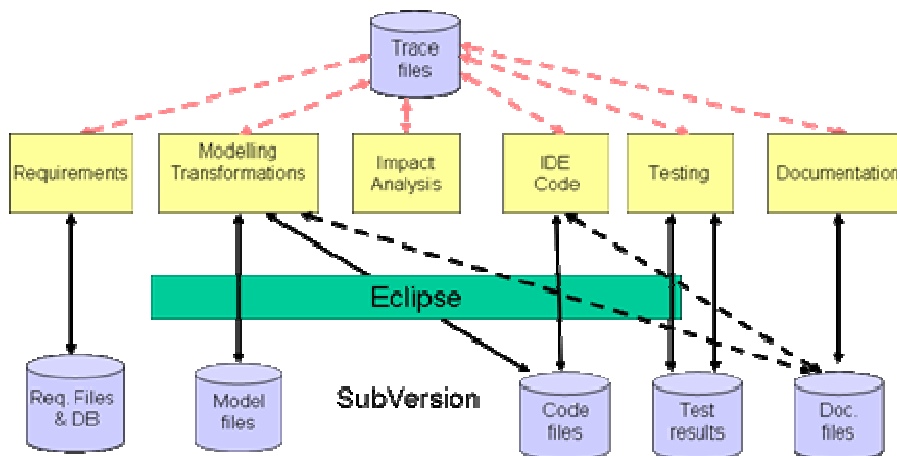


Fig. 1. Integrated toolsuite

A key issue of this tool suite is integration. The different tools should provide an integrated user experience. One simple measure of such can be how many different tools need to be started in order to perform a set of task. The different tools also need to be able to exchange information, either directly or through reading and writing files or other artifacts.

The Eclipse platform provides an excellent basis for such an integrated tool suite, and a lot of model related tools are now available as plugins for Eclipse. Ideally all tools in the MDD toolsuite should be Eclipse based.

Related to source code developers have become dependent of source code management systems. In MDD models are as important as code, and mechanisms providing SCM functionality such as versioning, diff and merge on models is extremely important. Currently not many tools exist that provide this functionality.

The issue of low cost is mentioned since over the later years one has become accustom to powerful tools being available as Open Source, again Eclipse is a good example.

2.1.2. Tool Substitutability and Standards

Another issue on the long term is tool substitutability. It is probable that at some point in time tools in the tool suite will be changed for new tools with better functionality, or that one user prefers a different tool than another. The example of changing what tool is used for UML modeling is representative. Even if there is no immediate need

to change a tool, the possibility of doing so is an important strength. The presence of substitutable tools also creates an environment where healthy competition will emerge between tool vendors; this will in turn benefit the end users of the tools.

In MDD a lot of the value will be placed in the MDD specific artifacts such as profiles, transformations and in the models themselves. When tools are substituted it is essential that this value is not lost, meaning that the artifacts still need to be usable by the new tools. In this respect standards are essential.

The Object Management Group (OMG) has defined a number of standards related to MDD, but not all of these are implemented commonly among tools. Even for tools that do implement the standards, interoperability is not always straight forward. The task of trying to move UML models with diagrams from one UML tool to another is a typical example of this. In this case standards exist for both the model information (defined by the XMI standard [1]) and the diagram information (Diagram Interchange Format specification [2]), the latter is implemented by very few UML tools and the former is a larger issue. The problem with the former is that there exists a number of versions of XMI and a number of versions of UML, meaning that one tool may implement XMI 1.4 for UML 2.0 while another implements XMI 2.0 for UML 2.0. Most commercial tools have extensive import options for handling these different formats, but again the diagram information is largely non-interchangeable.

This is actually quite an issue as the diagrams themselves contain a lot of valuable information from the person who created the model. This information may not be essential to the MDD toolchain, but is needed by human users of the models.

Another issue regarding substitutability of UML tools is that of profiles. Moving a profile from one tool to another and moving models with profiles applied is a clear challenge today. Since the use of profiles is essential in MDD this is an issue that needs to be handled.

In the area of model transformations the two key standards are QVT [3] (covering model to model transformations) and MOF Model to Text [4] (covering model to text transformations). There exist some implementations of QVT, but the leading model to model transformation tool, ATL¹, does not implement this standard. For model to text transformations few, if any, tools currently implement the relevant standard. Only when tools implement the standard it will be possible for the standards to mature.

2.1.3. End to End Traceability

Since artifacts in a MDD process typically are interrelated, either by one model being generated from another or otherwise, the issue of keeping track of this information and making use of it is essential. This is normally referred to as *traceability*.

What is needed is end-to-end traceability. This means that trace links should be kept from requirements through use cases, analysis and design models all the way down to the generated source code. Having this information extracted and stored automatically and transparently is necessary, but the real value is in how this information can be utilized in the development process.

This information can be used for multiple purposes. The most evident is that of some kind of impact analysis. For instance: if a requirement changes the trace information can be used to find and alter the other elements that need to be updated. In the other direction one could envision that a failed test could be linked back to the

¹ Atlas Transformation Language, <http://www.eclipse.org/m2m/atl/>

code it tested and then back again to the model elements that were the source of the code.

Traceability information is also useful in evaluation of requirements and test coverage. If design elements are connected to their requirements a simple query can present what requirements have not been brought into design. The same goes for tests, if tests are connected to the elements in the model or code they are to test one can discover what parts of the system do not have tests implemented.

It should be noted that it is equally important to maintain trace information about the MDD specific artifacts such as transformation descriptions. This means that one will be able to track down what part of a transformation created a part of the source code from a part of a model. This can be used to identify errors in the transformations themselves.

2.1.4. The ability to handle large and complex models

A model, by definition is a representation of an abstraction of something. But, for non-trivial systems, even the abstraction becomes quite complex. It is key that tools support the developers in handling this complexity.

2.1.4.1. Modelling at different levels of abstraction

In order to handle complex models it is important to be able to move between different levels of abstraction in a “seamless” manner. An analogy to this is Google Earth, here one can start out in space and zoom in on the parts of the world one wants to examine, the closer one looks the more detail is added to the view. The following presents an example of the wanted functionality:

1. The Architect creates a high level architecture model of the system. This might be a pure logical model. (See **Fig. 2** top).
2. The Architect then refines the model by further detailing one or more of the logical components. This could be done by double clicking on the core class which will open up a view of the inside. (See **Fig. 2** middle). First time this will be an empty view showing only the external connectors.
3. The Architect/Developer then refines the model by adding more elements to the view and connecting them to the external “ports”. (See **Fig. 2** bottom). This view could of course also contain abstract elements that could be further detailed.

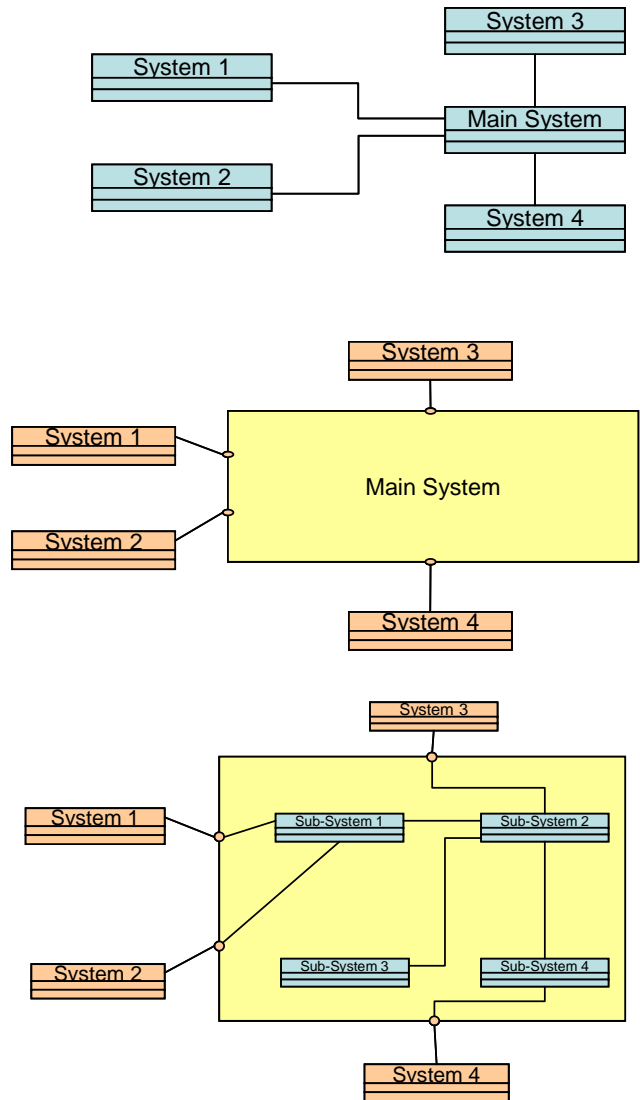


Fig. 2. Different abstraction levels

In modelling this is an issue that is shared between the modelling language and the tool providing the functionality to model using the language. In UML some of the needed functionality can be obtained using composite structure diagrams, but not all tools support this in the same way. In fact the support for Composite Structure diagrams varies much, Rational Software Architect is close, albeit not complete, in providing what we need.

2.1.4.2. *Different views and model synchronization*

One needs to be able to create models at different levels of abstraction and still be able to keep them connected or related. One example is the application of different profiles to the same Platform Independent Model (PIM). A possible scenario is described in the following and depicted in **Fig. 3**.

1. The Developer creates the Domain model containing the relevant domain entities (here called a PIM).
2. The Developer wants to apply a Reliability profile to the PIM model. To avoid polluting the original PIM model with profile stereotype information (we want to apply another profile later) we need to copy the PIM to PIM'. The copy process could be done with a copy transformation.
3. The Developer then applies the Reliability profile to the copy of the PIM model and then runs the transformation that will result in a platform specific model (PSM) or a code model.
4. The Developer sees that something is missing from the original Domain model (PIM) and adds new information to that model. The tool suite should then propagate the changes into the PIM' model and rerun the transformation. The Developer now wants to apply a Persistence profile to the PIM. The Developer runs a copy transformation creating PIM''.
5. The Developer then applies the Persistence profile to the copy of the PIM Spread model and then run the transformation that will results in a platform specific model (PSM) or a code model.
6. The same steps are repeated in applying the profile for Testing.

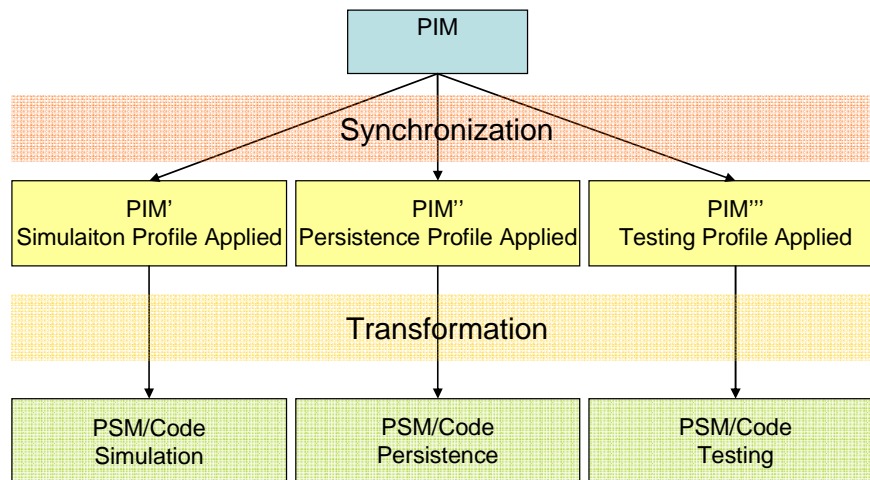


Fig. 3. Model Synchronization

The scenario described above suggests that the developer himself has to run the different transformations in order to create new copies etc. Ideally this would be hidden by the tool. For instance the user could turn on and off profiles in a view of the model (like features in a graphical information system).

2.1.5. Tool usability

In model based system development one can see two sets of artefacts; one directly related to the system being developed and one containing the extra MDD artefacts such as profiles, DSLs and transformations. Related to these two sets one can envision two different roles; the software developer and the MDD developer. As discussed in [5] the development of the MDD artefacts also is a development process. At a high level the time spent in this process should be balanced in saved time or higher quality in the software development process.

For the MDD developer it is important that the tools and technologies she is using for defining profiles, languages, transformations etc are as easy to use as possible. If one looks at languages for defining transformations, these are usually “new” languages and thus need to be learned. Nevertheless the languages should be as simple as possible to learn. Today developers have become used to have strong tool support for standard programming languages, such as code completion. Tools for transformation definition also need to have this support. In honesty this is present in some of the available tools today.

When the MDD artefacts are ready it should be a simple task for the MDD developer to deploy the artefacts to the development team. Ideally this should be doable within minutes.

Once deployed the developers should not have to see the internals of the MDD artefacts. Transformation descriptions are a good example of such an artefact, when deployed with the right metamodel it should be possible to invoke the transformation through suitable menus in the integrated tool. The details of the transformation code are not necessarily of interest to the developer. Currently most tools seem to be developed with the MDD developer in mind, and not the end user of the MDD artefacts.

In addition developers today are used to get rapid feedback when they are developing. Tools such as Eclipse and IntelliJ provide instant feedback if illegal Java statements are entered in the editor. Based on experience this feature seems to be of significant importance to productivity. Similar functionality should be present in the MDD tools.

2.2. Value Added Technologies

2.2.1. Model based simulation

For WesternGeco it is important to be able to simulate various aspects of systems as early as possible. For instance given a requirement of availability for a large equipment configuration one needs to break down the configuration into its parts and predict what availability is needed from the parts.

There currently exist a number of tools that can be used to create such simulations. Typically one creates some kind of model (mathematical or other) of the things that need to be simulated. These models do not resemble the models one is used to see when it comes to system development.

Ideally one should be able to use models of the system created for system development (these will be created anyway) as input to simulation tools. One possibility is that system development models are annotated with various aspects

relevant for simulation. Examples of such may be capacities, latency, meantime before failure etc (depending on the case at hand). The model and its annotations can then be transformed into a format that is used by the simulation tool to perform the actual simulation.

There already exists ways of annotating models with parts of this information, one of such is the UML profile for QoS[6]. The transformations toward the simulation tools will have to be developed.

2.2.2. Model based testing

Testing is an important part of software development. In a MDD context it would be nice if tests for the system could be generated based on the system development models. One issues is performing tests on the models them selves, but we believe that there is a potential in being able to generate tests for the resulting systems. In essence this means generating tests built to test the system that is also being generated from models.

Unit tests are a commonly used mechanism to perform tests during development. If models are annotated with pre and post conditions it will be possible to generate such tests from the models. In this one will generate the tests toward a test framework such as JUnit.

MDD techniques can also be used for black box testing through generation of “emulators” of external parts of the system. These emulators are to provide input data and behaviour that makes it possible to test the component at hand. This is highly relevant in the development of large systems as all components will not be ready at the same time, and there is a need to perform integration testing as early as possible. In this it is important to have ways of modelling the essential characteristics of these external components without creating a complete and detailed implementation.

2.2.3. Knowledge Discovery

Knowledge discovery is a term used about understanding existing systems. Related to MDD one key element is Reverse Engineering, creating models from existing systems, usually by automatic analysis of the source code of the system.

Many tools today provide the ability to reverse engineer code (for instance Java code) to UML models. It is our experience that, for any non-trivial system, the resulting models are of little use to really understand the system as they are a UML representation of a Java program (Java in pictures). One example is that in order to implement a UML association with cardinality larger than one, one will for instance use a `java.util.Vector`, that will contain the instances of the type pointed to by the association. Many reverse engineering tools will create a UML class for the association source with an association to `java.util.Vector`. The information of the type actually pointed to is either lost or is not obvious. In fact the resulting UML model has `java.util.Vector` as one of its key classes, which obviously is not right.

What is needed is the possibility to tune the way the models are created from the code with the goal of creating models that are useful. Such technologies are currently being worked on in the MODELPLEX project.

Another related aspect is that of being able to understand complex models, regardless of their origin. One example would be the reverse engineered Java code mentioned. A user should be able to create or get views of the model suitable for the purpose.

3. The importance of Open Source

The commercial front-runners in the MDD tooling market, such as Borland and IBM Rational, provide sets of systems that remedy some of the issues mentioned in this paper. However these suites are quite costly.

As mentioned the Eclipse platform and the different plugins related to modelling provide a lot of needed functionality as open source. Indeed the above mentioned commercial actors have contributed greatly to this. But more importantly this platform has become a basis for smaller vendors and universities to provide usable solutions. It also seems clear that the activity in open source on the Eclipse platform contributes to the commercial solutions provided.

The value of open source for an end user like WesternGeco is not only that the tools are free (“purchase time”), but the possibility to alter the solutions and to integrate them with other tools is equally important. It is also easier to start experimenting with something that is freely available on the Internet. Through such experimentation, at ones own pace, it will be possible to get hands on ideas of how such technologies can be used in ones company.

4. Future work

As mentioned in previous sections of this paper there is work going on in most of the technology areas presented, both inside and outside of the MODELPLEX project. WesternGeco will cooperate closely with the technology providers in MODELPLEX to ensure that the provided solutions meet the requirements. There currently exists a plan of how the different results are to be evaluated by WesternGeco and the other end users in the project.

In 2006, as part of the MODELWARE project, WesternGeco and SINTEF carried out a controlled experiment [7] to investigate whether the use of MDD provided higher productivity than a traditional approach. The experiment concluded that it, based on the results, could not be proven that there was a significant difference between the two approaches. A similar experiment, comparing MDD using MODELPLEX technologies and a traditional approach will be performed toward the end of the MODELPLEX project.

5. Conclusions

We believe that the MDD paradigm has great potential but tools and technologies need to become more robust and mature before the whole potential is achievable in an industrial setting. We would like to re-emphasise, that having an integrated flexible MDD environment with end to end traceability is essential. The ability to derive tests directly from them models is also needed. Such capabilities should be available at a low cost. Currently well integrated suites are too costly and in most cases do not provide the wanted substitutability that is important to ensure keeping the value of created artefacts.

In this way will be possible to infuse MDD in larger parts of the software development process, thus providing the potential of harvesting the MDD promise of improved productivity and quality.

Acknowledgement: This work has been co-funded by the European Commission within the 6th Framework Programme project MODELPLEX contract number 034081 (cf. <http://www.modelplex.org>).

6. References

1. OMG: MOF 2.0/XMI Mapping, Version 2.1.1. Object Management Group (2007)
2. OMG: Diagram Interchange Specification, v1.0. Object Management Group (2006)
3. OMG: Meta Object Facility (MOF) 2.0 Query/View/Transformation, v1.0 (2008)
4. OMG: MOF Model to Text Transformation Language (MOFM2T), 1.0. Object Management Group (2008)
5. Gavras, A., Belaunde, M., Pires, L.F., Almeida, J.P.A.: Towards an MDA-based development methodology for distributed applications. First European Workshop on Software Architecture, Vol. Volume 3047/2004. Springer Berlin / Heidelberg, St. Andrews UK (2004) 230-240
6. OMG: UML Profile for Modeling QoS and Fault Tolerance Characteristics and Mechanisms. Object Management Group (2008)
7. Nordmoen, B., Neple, T.: Western Geco ROI, Assessment, and Feedback. MODELWARE project (2006)