REACT-MDD
Reactive Traceability in Model-Driven Development

Marco Costa
Universidade Autónoma de Lisboa, CESITI, Lisbon, Portugal
mbcc@acm.org

Alberto Rodrigues da Silva
INESC-ID / Instituto Superior Técnico, Lisbon, Portugal
alberto.silva@acm.org

Keywords: Traceability, Model-driven Development, Meta-model.

Abstract: The development of information systems has evolved to a complex task, regarding a multitude of programming and modelling paradigms, notations and technologies. Tools like integrated development environments (IDE), computer aided systems engineering (CASE) and relational database systems (RDBMS), among others, evolved to a reasonable state and are used do generate different types of artefacts needed in this context. ReacT-MDD is a traceability between artefacts open model that was instantiated in a prototype. We present and discuss some practical issues of ReacT-MDD in the context of reactive traceability, which is also described.

1 INTRODUCTION

The development of information systems have been changing regarding not only technologies but also notations and methodologies. As the complexity of the implemented systems is growing steadily, the need for ways of systematically develop applications increase. New levels of abstraction, heterogeneous environments, different programming paradigms and complex contexts, are just some issues the software developer has to deal with. The model-driven development (MDD), which has its roots on the methodologies boom of the 1970s and 1980s (Jackson, 75; Martin, 89) is promising to increase the productivity of the development and maintenance tasks. MDD is being helped by tools like integrated development environments (IDE), computer aided systems engineering (CASE) and relational database systems (RDBMS), among others, which are already reasonable evolved and are used do generate different types of artefacts. An artefact may be considered as something that is produced or crafted in the context of some tool, not just a data file (e.g., a Table in a RDBMS, a Class written with an object oriented programming language, a business requirement written in plain text). An information system involves a set of active artefacts that cooperate towards a common goal. These artefacts are present in multiple views, regarding the abstract layer we take as a viewpoint. For instance, a system may be described by its functions, data structures and technologies, among other features. Documentation is part of the solution, just as the formulation of a problem may be considered part of its solution. From requirements (Palo, 2003) to code (Costa, et al., 2007), it is possible to trace all artefacts of the system, adding new ones, named *traces*. Development of new applications and maintenance of the existing ones should be accompanied by tools and methodologies which minimize the risk of introducing a state of incoherence between some artefacts. When this problem is not tackled the reality shows that programming code evolves with no or minor relation with models (Figure 1).

In large applications, when models are almost completely outdated the system is in risk of becoming unmaintainable. *Traceability*, as a generic term, is used in many different contexts from food industry to software development and maintenance. Traceability deals with keeping records of relations between artefacts of the same, or different abstract levels. We propose the term *reactive traceability* as

a characteristic of a system that not only keeps information of the relations, but also can react to and prevent changes to artefacts or to its relations (traces).

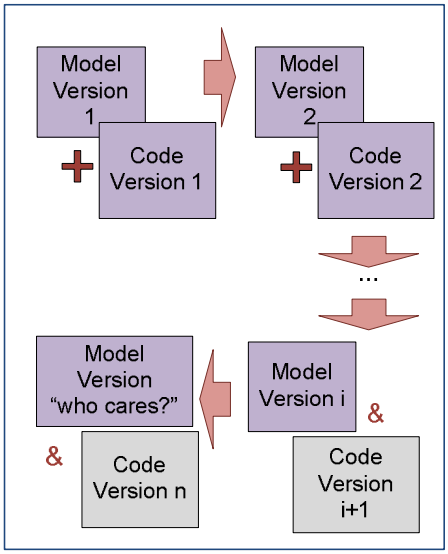

Figure 1. Relation between models and code as time goes by becomes less reliable.

Automated transformation of models-to-models, models-to-code, as well as code-to-models, is becoming a reality. These transformations generate output artefacts from source artefacts. The relation between input and output artefacts of a transformation is just one, and obvious, type of semantic relations among others. Object Management Group QVT (Queries, Views and Tranformations) (OMG, 2005) is aimed to standardize not only transformations with models but also other operations with models, like queries and views. Our approach takes QVT as a starting point to accomplish the construction of automated transformations between models and implements a way of maintaining traces between artefacts (models and code) as well as reacting to changes for the sake of system coherence. Reactive traceability is a way to accomplish the maintenance of coherence between artefacts with or without human intervention.Our proposal is aligned with OMG standards and recommendation like UML (OMG, 2010) and QVT. Even if QVT by itself has not a complete implementation, it is however a starting point to different approaches and products like Tata ModelMorf (Tata, 2007) or Compuware Optimal J (Compuware, 2010). Other concurrent approaches to QVT include ATL (Eclipse, 2010), Mistral (Kurtev, et al. 2005) and EML (Kolovos, et.al., 2006), Executable UML (Mellor et al., 2002). The vision of traceability in a MDD perspective is shared with other initiatives, such as those in (Aizenbud-Reshef, et al., 2006; Walderhaug, 2006; Oldevik, et al. 2006). Also, the ReacT-MDD vision uses meta-modelling and model-based model conformance (Paige et al., 2007).

In Section 2 this paper explains some general concepts that are inherent to traceability. Section 3 introduces a reactive traceability conceptual model. A prototype was developed that validated the ReacT-MDD approach (ReacT-Workbench) and enhanced his scope. In Section 4 are discussed some practical issues like implementation decisions that were made in this working prototype. Section 5 states some relevant conclusions of this work.

2 TRACEABILITY VIEWPOINTS

Transformations between models or between models and code (usually from models to code and not the opposite) are a relevant issue to reactive traceability. After transformations are performed traces are created (implicitly or explicitly) but this is not the only action that can create traces. There are two viewpoints (Figure 2) to the issue of creating traces: a) in legacy applications one may consider artefacts already existent and traces will instantiate and describe some implicit or explicit semantic relations, or dependencies, between them; b) from a starting point in an artefact (diagram or code) it is necessary to create (or generate) another artefact, using some type of transformation.. These two viewpoints are both necessary in a solution that implements reactive traceability. The first item has a focus on creating and maintaining the semantic relations between existing artefacts and the word *existing* is a keyword to understand this viewpoint. At some time of the development process changes in the system will become evolutions of an existing state. For each change the system coherence is checked and decisions are made about the new achieved system state. The second gives more importance to the generation process (Herrington, 2003; Dollard, 2004) and traces are relations between old and new (generated) artefacts. The second viewpoint is related to the IEEE traceability definition (IEEE, 1990): “the degree to which a relationship can be established between two or more products of the development process, especially products having a predecessor-successor or master-subordinate relationship to one another”. When a generation of a set of artefacts occurs it is necessary to record the

new dependency relations that are created in the process. These two approaches are just not only valid but necessary, in a development process and reactive traceability must include both.

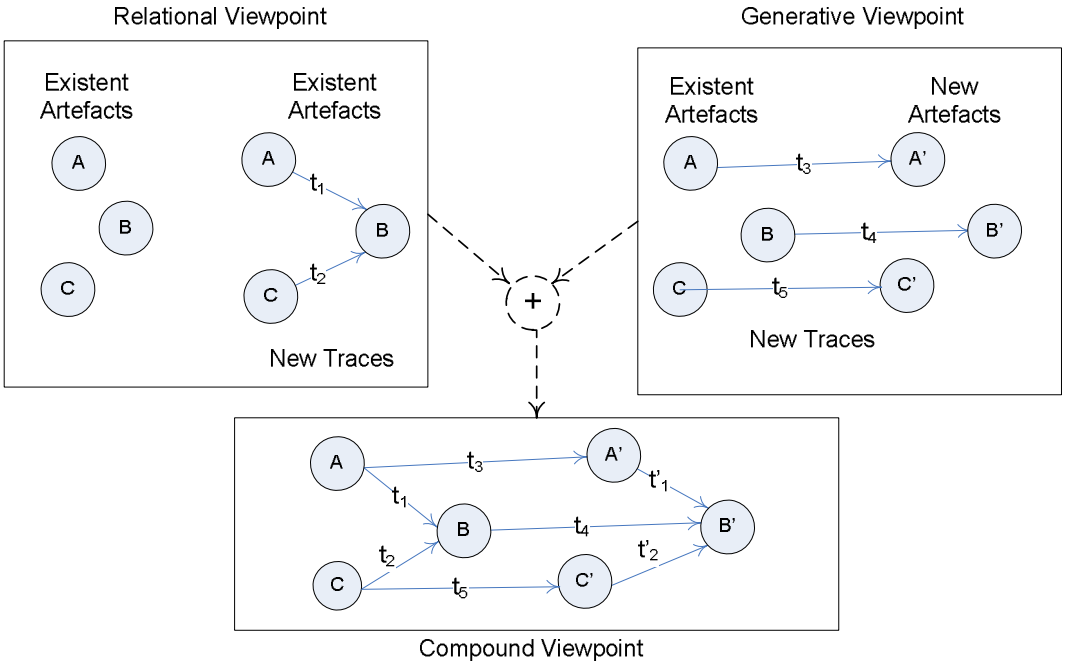

Figure 2. Artefact relation viewpoints

3 A REACTIVE TRACEABILITY MODEL

The building blocks of traceability, in any system, are traces and artefacts. As an extension we may consider that traces are also artefacts. With that in mind it is possible to consider even traces between traces. A trace may be considered as an algebric relation between elements (which in this context are traces). However, traces and artefacts are not enough to define a traceability model (Figure 3). ReacT-MDD approach defines an event model that starts events, when some action over a given artefact is performed. The event may trigger a synchronization decision (e.g., "Change a name of a Java Class when a given UML class name has changed?") with or without user intervention. Also, an artefact is described by a meta-model which can be more or less rich, in relation with an implementation concept known as *plug-in* which enables ReacT-Workbench to act over artefacts of different applications.

The meta-model is defined for any kind of artefact that is to be included the in traceability solution. It is a description needed for an artefact to be included in the trace definition.

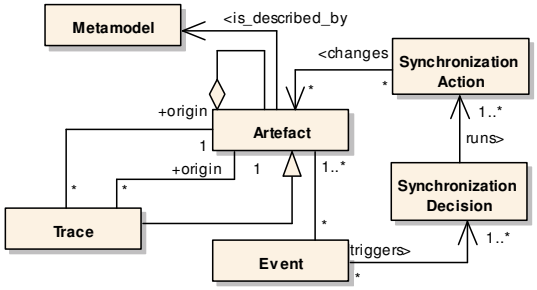

Figure 3. Simplified UML class diagram of reactive traceability.

Traces may be defined between occurrences in particular or between types of artefacts, given a context. For example, in some project, it is possible to define a trace between the business class "Entity" and the Java class "Person". When a change is made to one of this artefacts, ReacT-MDD triggers an event that require one or more synchronization decisions to be made. In the same example: "1 – Leave the artefacts unchanged", "2 – Renew traces

with new values", "3 – Delete trace". It is possible also to define traces between groups of artefacts like "UMLClass : BusinessModelX" or "CSharpClass : AppTestX". In that case, the traces are verified for each artefact that conforms with the meta-model definition and is included in the specified context.

4 A PRACTICAL APPROACH

ReacT-MDD was instantiated in a prototype (ReacT-Workbench) that implements the concepts and delivers the basis for researching in the practical issues of this field. The tool was developed with scalability requisites that concern the fact that it must deal with artefacts generated by a large number of tools. The approach was to develop several test plug-in modules that give the ability to interact with each needed tool or artefact. Also each plug-in gives an access to a meta-model of the targeted artefacts (Figure 4). The tool was developed in the C# programming language and has a plug-in for the same language. It was defined also a traceability language (which is an extension to QVT) which is needed to define a trace between artefacts. Also it was defined a context definition module that links each real occurrence of the artefact with its representation.

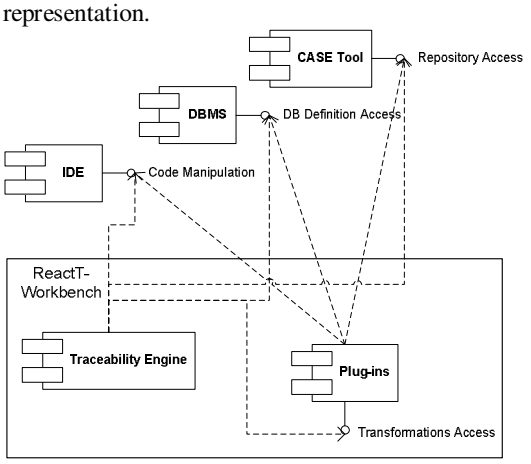

Figure 4. UML component diagram of React-MDD interaction with external applications.

When a plug-in is available for some application, or for some kind of artefacts (e.g., Microsoft Excel files) the ReacT-Workbench gets access to the corresponding meta-model and associated concepts (in the same example, *Workbook, Column, Row, Cell)*. These concepts may be refered in all traces that are defined after.

Of course, more components of different types may be added if they produce or consume artefacts with traces. The link between the Traceability Engine and each other component varies from the type of component. Each tool to be included in the traceability solution must have the plug-in that will be used to access the relevant artefacts which have traces.

The traceability solution, which is an implementation of ReacT-MDD (e.g., ReacT-Workbench), plus all artefacts and plug-ins, is designed to work on development operation environments, as opposed to test and production operation environments. As it acts with the structure of the target applications, if the structure is not changed there is no need to control traceability. This operation requisite ensures that changes to documentation artefacts of the project should be made in the same operation environment of code, models, or others. The operation environment may be just one or a set of logically connected operation environments as well. This is achieved at the system level (e.g., with drive mapping).

The ReacT-Workbench implemented a Traceability Engine (TE) which permanently verifies the coherence state of the system in a three phase approach:

Phase 1: The TE polls each artefact. When a change is made to an artefact, the TE adds that artefact to a list of changed artefacts. This phase is possible to be achieved concurrently, given a set of artefacts, but Phase2 can't start until all artefacts where polled. For each type of artefact (e.g., code, model, text document) this phase may involve parsing and constructing an object tree graph and compare each node with a corresponding one in a stored representation of the initial artefact.

Phase 2: For each artefact in the list of changed artefacts TE searches in an artefact event list all the relevant events related to that artefact. For each of these events found, it is necessary to verify if a valid action was made in the system. Possible valid actions are: create, update and delete. The read action is not valid as it does not change the system state. The create action is valid when used with meta-model artefacts. Only then it is possible to say if an event related directly to the artefact exists. If this is the case, the event is triggered.

Phase 3: After the artefact events are verified it is necessary to check if traces still hold. For each artefact in the list of changed artefacts TE recursively searches trace events that refer artefacts contained in the artefact. If a trace event is founded,

and the relevant artefact and action hold, the event is triggered (Figure 5).

We consider a complete 3-phase cycle as a *traceability event cicle,* which is scheduled: a) at a moment, present or future, which is more convenient to the user, b) periodically with a time interval (from ms to days).

When the trigger conditions are satisfied TE runs synchronization decisions. When the user selects one of the decisions available TE runs a set of synchronization actions associated to that decision. After the decision is taken the system is in equal or more coherent state than before (i.e., has the same number or less artefacts in an incoherent state). If a decision of solving a coherence issue is postponed, a warning is generated and logged in a *to-do list* for further processing. Also this incoherence is archived to override future verifications in Phase 2.

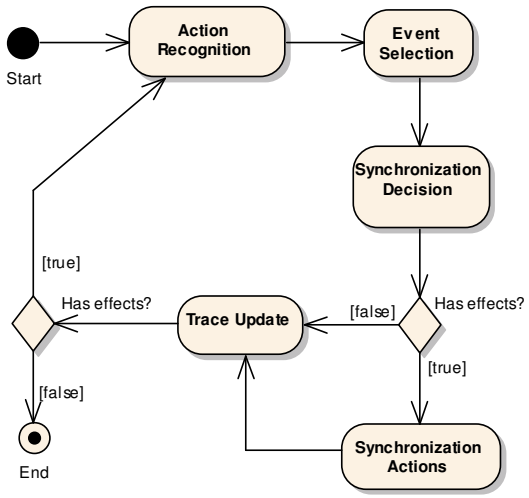

Figure 5. Event recognition and triggering of synchronization actions.

The user interaction with the solution is minimized if trace and artefact events have only one synchronization decision (representing just one option). In that case TE may enforce or omit the related actions (in this case generating an entry in the to-do list).

ReacT-Workbench is also able to generate some metrics derived from the meta-model concepts (e.g., how many classes, attributes and associations exist in a particular UML model, or how many tables exist in a relational database).

Future development is needed in the design of attractive and efficient graphical user interfaces to deal with issues like simulation of action effects, tests, completeness and terminating analysis.

5 CONCLUSION

There is a variety of development tools from different types (e.g., CASE , IDE) that are being used all over the world. Each tool is generating artefacts that, in many cases, should be coherent with other artefacts crafted with other tools. The number of artefacts is growing and the relations between them should be maintained along the project's lifecycle. Notations have been replaced, programming languages and methodologies have evolved significantly but, at large extent, users are still responsible for the maintenance of the coherence between artefacts. Existent tools can do some kind of synchronization but the ways for achieving this are tool dependent and with difficult customization.

The convergence of modelling notations to UML was an important factor because it gives some stability to this field. Development teams are still adopting UML as the notation and practical issues are emerging with more experience and new releases of the standard. ReacT-MDD is an effort to define an open model to define tools that can deal with traceability between artefacts of a project, as well as react to changes in the state of coherence of the system.

The ReacT-MDD approach was instantiated in a prototype that deals with the explained concepts. Instead of developing another tool as a plug-in for an existent single tool or set of tools, the ReacT-MDD defines an open architecture that enables different industry providers to be included. Also, it is orthogonal to operating systems, tools, development environments, methodologies and languages.

This work presented some issues that where considered in the implementation of a solution prototype. Traceability is still regarded has a documentation activity. If the human resources involved in the project are not conscientious about the relevance of documentation in maintenance phases, traceability is not seen as a critical issue.

In our perspective, traceability is not about documentation of the system, it is about the system itself, its parts and the way they are related to each other. Reactive traceability is a proposal driven by that principle. ReacT-MDD, as a *reactive* solution it is supposed to act, creating, updating or deleting artefacts of the target system, in *reaction* to an event. The number, and importance, of available artefact types and the level at which the tool is capable of interact with each of them is an important measure of its capabilities.

REFERENCES

Aizenbud-Reshef, N., Nolan, B. T., Rubin, J., Shaham-Gafni, Y.. 2006. Model Traceability. In *IBM Systems Journal, Vol. 45, Nr. 3*

Compuware, 2010. Optimal J. In *www.compuware.com.* Compuware.

Costa, M., Silva, A. R., 2007. Synchronization Issues in UML Models. In 9th *International Conference on Enterprise Information Systems*, *Funchal - Portugal*

Dollard, K.,2004. *Code Generation In Microsoft .NET*. Apress

Eclipse, 2010. ATL. In www.eclipse.org/m2m/atl/

Herrington, J., 2003. *Code Generation In Action*, Manning Pub. Co

IEEE, 1990. *IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries*. Institute of Electrical and Electronics Engineers

Jackson, M., 1975. *Principles of Program Design,* Academic Press.

Kolovos, D.S., Paige, R. F., Polack, F. A. C., 2006. On-Demand Merging of Traceability Links with Models. In *3rd ECMDA Traceability Workshop*

Kurtev, I., Berg, K., 2005. MISTRAL: A Language for Model Transformations in the MOF Meta-modeling Architecture, *Lecture Notes in Computer Science,* Springer (2005).

Martin, J., 1989. *Information Engineering: Introduction,* Prentice Hall

Mellor, S., Balcer, M., 2002. *Executable UML.* Addison-Wesley.

Oldevik, J., Neple, T.2006. Traceability in Model to Text Transformations. In *3rd ECMDA Traceability Workshop*

OMG, 2005. MOF QVT Final Adopted Specification. In *http://www.omg.org/docs/ptc/05-11-01.pdf,* Object Management Group

OMG, 2010. OMG Unified Modeling Language Infrastructure. In *www.uml.org/#UML2.0,* Object Management Group

Paige, R., Brooke, P., Ostroff, J., 2007. Meta-model-based model conformance and multiview consistency checking. In *ACM Transactions on Software Engineering and Methodology (TOSEM)* Vol. 16, Issue 3 (July 2007)

Palo, M., 2003. Requirements Traceability. In *Seminar Report, Department of Computer Science*. University of Helsinki

Tata, 2007. ModelMorf – A Model Transformer. In *www.tcs-trddc.com/ModelMorf/ index.htm*. Tata

Walderhaug, S., Johansen, U., Stav, E., Aagedal, J., 2006. Towards a Generic Solution for Traceability in MDD. In *3rd ECMDA Traceability Workshop*