Survey of Traceability Approaches in Model-Driven Engineering

Ismênia Galvão and Arda Goknil
Department of Computer Science
University of Twente
P.O. Box 217, 7500 AE Enschede, The Netherlands
{i.galvao, a.goknil}@ewi.utwente.nl

Abstract

Models have been used in various engineering fields to help managing complexity and represent information in different abstraction levels, according to specific notations and stakeholder's viewpoints. Model-Driven Engineering (MDE) gives the basic principles for the use of models as primary artefacts throughout the software development phases and presents characteristics that simplify the engineering of software in various domains, such as Enterprise Computing Systems. Hence, for its successful application, MDE processes must consider traceability practices. They help the understanding, capturing, tracking and verification of software artefacts and their relationships and dependencies with other artefacts during the software life-cycle. In this survey, we discuss the state-of-the-art in traceability approaches in MDE and assess them with respect to five general comparison criteria: representation, mapping, scalability, change impact analysis and tool support. As a complementary result, we have identified some open issues that can be better explored by traceability in MDE.

1 Introduction

Models have been used in various engineering fields to help managing complexity and represent information in different abstraction levels, according to specific notations and stakeholder's viewpoints. A model is a symbolic system expressed in a language [22]. Each kind of model is represented by an appropriated modelling language and can be applied to certain purposes [25]. In Software Engineering, various models can be used for representing software artefacts, according to the diverse development paradigms.

Model-Driven Engineering (MDE) gives the basic principles for the use of models as primary engineering artefacts throughout the software development life-cycle [3, 19]. A software system is specified as a set of models that are repeatedly refined until a model with enough details to implement the system is obtained [1]. Figure 1 illustrates model refinement steps in MDE, in which more abstract models are transformed into more detailed ones. Since all models are representations of the same system, every transformation step should preserve the intended meaning of the source model and eventually bring new details.

When applied in practice, the general outline of the MDE process should follow and address some stable general principles and scenarios of software development, such as separation of concerns, iterative development, refactoring or reverse engineering. Moreover, since the system is developed as a series of transformations over models, a change in one model must be propagated through the rest. The propagation may be in two directions: to models derived from the changed model and to models from which the changed model is derived.

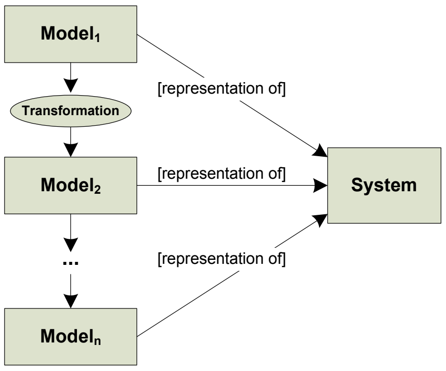

Figure 1. Refinement of models in MDE

Traceability is a necessary system characteristic as it supports software management, software evolution, and validation [24]. It is also fundamental on the definition of the results of change impacts. Traceability practices help on the understanding, capturing, tracking and verification of software artefacts and their relationships and dependencies with other artefacts during the software life-cycle. We believe that the established use of MDE approaches should explicitly include traceability support to provide more benefits on developing software for domains such as Enterprise Computing Systems.

An interesting related work is presented by Aizenbud-Reshef et al. in [1], where the authors review the most recent advances on technologies to automate traceability and discuss the potential role that model-driven development can play in this field. In addition, a survey on tracing approaches in traditional software engineering, and the elaboration of a traceability taxonomy, is presented in [26].

In this survey, we discuss the state-of-the-art in traceability approaches in MDE and appraise them with respect to five general comparison criteria: *representation*, *mapping*, *scalability*, *change impact analysis* and *tool support*. The evaluated approaches are distributed into three categories that range from the use of requirements as non-formal models to approaches that utilize models and metamodels according to the MDE paradigm. This research is meant as a starting point for identifying open issues that can be better explored by traceability in MDE.

The remainder of this paper is organized as follows. In Section 2, we present an overview of the traceability approaches in MDE. In Section 3, we present an evaluation of the approaches according to the five criteria. In Section 4, we discuss some open issues on the development of model traceability frameworks. Finally in Section 5, we describe the conclusions of the paper.

2 Traceability Approaches in MDE

Traceability is the ability to establish degrees of relationship between two or more products of a development process, especially products having a predecessor-successor or master-subordinate relationship to one another [13]. In practice, traceability mechanisms help to identify the origin and rationale of software artefacts [27], as well as they provide essential assistance in understanding the relationships between these artefacts within and across software development stages.

For example, a Java class may be traced back to its design class, analysis class, and ultimately to the requirement that motivates its presence in the system. In the case of model transformations in model-driven development processes, a trace would relate elements in a source model to the generated elements in a target model. MDE provides new promising ways to automate the discovery and the generation of trace relationships [1], such as to trace artefacts over a chain of model transformations.

The traceability approaches that will be considered for analysis in this paper were classified into three categories: *requirements-driven approaches*, *modeling approaches* and *transformation approaches*. The requirements-driven approaches use requirements models as abstractions to guide their traceability methods. The modeling approaches are interested in how metamodels, models and/or conceptual frameworks are involved in tracing processes. Finally, transformation approaches make use of model transformation mechanisms for generating trace information.

2.1 Requirements-Driven Approaches

In the field of Requirements Engineering, Gotel and Finkelstein [14] define traceability as "the ability to describe and follow the life of a requirement, in both forward and backward specification, to its subsequent deployment and use, and through periods of ongoing refinement and iteration in any of these phases". Tracing requirements in both forward and backward directions helps stakeholders and developers to understand the semantics of requirements in more detail.

The following subsections present five approaches which use requirements models as important abstractions to guide their traceability methods.

2.1.1 Requirements Traceability and Transformation Conformance (RTTC)

In [4], Almeida et al. aim at simplifying the management of relationships between requirements and various design artefacts. They propose a framework which is a basis for tracing requirements, assessing the quality of model transformation specifications, metamodels, models and realizations. The authors state that the most suitable traceability definition for their work is that "the means whereby software producers can 'prove' to their client that: the requirements have been understood; the product will fully comply with the requirements; and the product does not exhibit any unnecessary feature or functionality". The methodological framework they propose allows designers to relate the requirements in the early stage of the development to the various products of the model-driven design process.

Traceability cross-tables are used for representing relationships between application requirements and models, considering different model granularities and also the identification of conformant transformation specifications. Since model-driven techniques consist of different abstraction levels, like platform-independence and platform-specific, they propose a notion of conformance between models to trace requirements throughout these different levels. They also formulate the notion of satisfaction of requirements by models that are produced by transformation chains but they deferred the change impact analysis of requirements over these models to future work.

2.1.2 Event Based Traceability (EBT)

Event-Based Traceability (EBT) is a method for automating trace link generation and maintenance. Cleland-Huang, Chang and Christensen [5] present an interesting study for requirements traceability, which uses EBT for managing evolutionary change. In this method, requirements and other traceable artefacts, such as design models, are no longer directly related, but linked through publish-subscribe relationships. This mechanism is based on the Observer design pattern [12]. Instead of establishing direct and tight coupled links between requirements and dependent entities, links are established through an event service. First, all artefacts are registered to the event server by their subscriber manager. The requirements manager uses its event recognition algorithm to handle the updates in the requirements document and to publish these changes as event to the event server. The event server manages some links between the requirement and its dependent artefacts by using some information retrieval algorithms.

The main components of the system are the event server, the requirements manager and the subscriber manager. The requirements manager handles the requirements and is responsible for triggering change events as they occur. The event server is primarily responsible for handling subscriptions, receiving change notifications, and forwarding customized event messages to the subscriber managers of dependent artefacts. The subscriber manager is responsible for receiving event notifications and handling them in a manner appropriate to both the artefact being managed and the type of message received. These messages carry structural and semantic information concerning the change context.

2.1.3 Goal Centric Traceability (GCT)

In [7], Cleland-Huang et al. introduce a goal-centric approach for managing the impact of change upon the non-functional requirements of a software system. Goal Centric Traceability (GCT) models non-functional requirements and their dependencies using a Softgoal Interdependency Graph (SIG). GCT enables developers to understand and assess the impact of functional changes upon non-functional requirements to maintain the quality of the system.

The process has four phases to analyze and to update the changes on dependent artefacts: goal modeling, impact detection, goal analysis, and decision making. In goal modeling, goals are decomposed into subgoals to reflect the fact that extensive interdependencies exist between various non-functional requirements (represented by softgoals). To understand the trade-offs among non-functional requirements, the subgoals are decomposed into operationalizations which provide candidate solutions for the goal.

During the impact detection phase, when a change occurs in non-functional requirements, a probabilistic retrieval algorithm dynamically returns links between impacted classes and elements in the SIG. In the goal analysis phase the user modifies the contributions, from the impacted goal elements to their parents. For each impacted element, changes are propagated throughout the SIG to identify potentially impacted goals. In the decision making phase it is decided to proceed with which proposed change. Stakeholders evaluate the impact of the proposed change upon non-functional requirement goals and manage risks.

2.1.4 Event Based Traceability with Design Patterns (EBT-DP)

In [6], Cleland-Huang and Schmelzer introduce another requirements-driven traceability approach. Their work builds on EBT [5] but describes a different process for dynamically tracing non-functional requirements to design patterns. This process is divided into two phases.

During the initial phase, user-defined traceability links are established. Instead of linking every model element in the design model to a non-functional requirement in a SIG, the elements are linked to a cluster which defines the design pattern. Then, a traceability link is established between the non-functional requirement and the cluster. This decreases the number of links established between design artefacts and non-functional requirements. In the second phase, the well established descriptions and invariant rules of a design pattern permit the automatic and dynamic generation of code during runtime (from the pattern to specific class implementations). As a consequence, implicit fine-grained links can be generated automatically. This characteristic increases the maintainability and the expressiveness of the method.

The authors explore the use of both dynamic and static generated traceability links, as well as the use of design patterns as intermediary models, to facilitate the traceability of non-functional requirements across the software development life-cycle. Therefore, the determination of when a non-functional requirement might be fulfilled by means of a design pattern is a non-trivial task which requires a good identification method.

2.1.5 Reference Models for Requirements Traceability (RMRT)

Ramesh and Jarke [28] follow an empirical approach and focus interviews conducted in software organizations to study a wide range of traceability practices. As a result of this work, the authors constitute reference models that include the most important kinds of traceability links for various software development elements. The trace entities and links for these models reflect the needs of real users.

One of the main motivations behind this study is to capture traceability needs of different stakeholders and present different reference models for these different needs. Their empirical study characterizes the participants as high-end and low-end users of traceability practices. The authors present trace models to reflect the trace entities captured by high-end and low-end users, and then customize a set of five reference models. Requirements are considered as traceable entities in all these reference models.

The reference model for low-end users is composed of four elements (requirements, compliance verification procedures, system components and external systems) which are interrelated by links that describe satisfaction, derivation, dependencies, and so on. The high-end use of traceability employs richer models. Four reference models were identified for this case: a requirements management submodel, a rationale submodel, a design allocation submodel and a compliance verification submodel.

2.2 Modelling Approaches

In MDE, trace metamodels are crucial to store and represent trace information, derived from the relations between source and target elements, explicitly as trace models. As an instance, the UML profile mechanism gives a solution to store and represent trace data. There is also a standard stereotype for traceability in UML [25]. It specifies a trace relationship between model elements or sets of model elements that represent the same concept in different models.

The modeling approaches we discuss in this section are interested in how metamodels, models and/or conceptual frameworks are involved in tracing processes.

2.2.1 Scenario Driven approach to Trace Dependency Analysis (SDTDA)

In Egyed [10], development artefacts are highly coupled and trace dependencies characterize the relationships among them abstractly. This study presents an automated approach for generating and validating trace dependencies. The main elements considered for the traceability analysis are: test scenarios, model elements (data-flow, use case and class diagrams) and implementation classes. These elements can be interrelated by different types of trace dependencies. The approach requires: an observable and executable software system; some list of development artefacts; scenarios describing test cases or usage scenarios for those development artefacts; and a set of initial hypothesized traces linking artefacts and scenarios. The main steps of the approach are trace generation and trace validation.

The behaviour of the system is observed using test scenarios. Executing those scenarios in the running system leads to observable traces that link scenarios to implementation classes or source code. The path between model elements and scenarios is reasoned in finding hypothesized traces. A footprint graph is built and manipulated via a set of rules in the automated trace analysis. The footprint graph is interpreted by traversing its nodes to elicit new trace information or to find contradictions in the result interpretation.

This study addresses the problem that the absence of trace information or the uncertainty of its correctness limits the usefulness of software models during software development. The proposed approach reduces the complexity of trace generation and validation by a set of test scenarios and hypothesized traces between the test scenarios and model elements. The runtime behaviour of these scenarios is translated into a footprint graph. The algorithm generates traces by using the rules that characterize how this graph relates to the existing hypothesized traces and the artefacts to which they are linked.

2.2.2 Operational Semantics for Traceability (OST)

Different types and representations of traceability exist with different characteristics and properties. Aizenbud-Reshef et al. [2] present an approach which defines an operational semantics for traceability in UML to capture different types of traceability and use a common notation in all situations. This also intends to provide a richer tool support for managing and monitoring traceability.

Three main issues for traceability are stated: querying (e.g. impact analysis, coverage queries); following links along the life-cycle of a project; and keeping the system and its documentation up to date. They define two types of semantics based on these issues: the *preventative semantics* and the *reactive semantics*. While preventative semantics describes things that should not happen, reactive semantics describes what should be triggered when something happens to one or more of the related elements or to the relationship itself.

The authors state that the operational semantics of a traceability relationship is defined by a set of one or more semantic properties. A semantic property of a relationship is a triplet *event, condition, actions*, where event involves an element of a relationship, condition is a logical constraint and actions can be either preventative or reactive actions. One of the observations of this study is that the notion of traceability that is of interest to a modeller should be captured using a set of semantic properties.

2.2.3 Unifiying Traceability Specification Scheme (UTSS)

Limon and Garbajosa [23] analyze several current traceability schemes and prove an initial approach for a traceability specification scheme based on this analysis to facilitate traceability specification for a given project, to improve traceability management, and to help automating some trace management processes. The following features are the starting point for the analysis and assessment done by these authors: process-related or product-related links; pre-requirements and post-requirements traceability relations categories; the traceability link purpose; and the items or objects to which the traceability link will relate.

The authors state that it is necessary to define an Unifying Traceability Specification Scheme, which contains the common features according to the analysis of the above listed features. According to the initial proposal presented by the authors for a Traceability Scheme Specification, the scheme should include the following items: a Traceability Link Dataset that will provide a wide basis to define traceability links; a Traceability Link Type Set; a minimal Set of Traceability Links; and a Metrics Set for the Minimal Set of Traceability Links.

2.2.4 Precise Transformation Traceability Metadata (PTTM)

Vanhooff and Berbers [31] define a UML profile based on a metamodel that gives support to transformation traceability links. Their approach allows the addition of semantically rich transformation traceability links into UML models, while keeping its consistency. These links enable transformation programs to explore the UML models and perform richer transformations. They may provide a better understanding on the effects of the model transformations. Such kind of information could be the history of model changes caused by transformations which were executed taking into consideration the model. As an instance, traces saved into the UML models can be used by future transformations to help on their own executions.

The authors list four important requirements that their transformation mechanism attend to. At first, the transformation traceability information should be left behind by all transformation units. Secondly, traceability links should be extended with transformation unit specific information. Another requirement is that all information should be kept in the UML model itself and, at last, it should be possible to easily add trace links manually for non-automatic transformations.

The main elements of their metamodel are transformation units, input and output elements, and element mappings. In this way, the profile conceptualizes dependencies between source and target elements, dependencies between a mapping and the transformation unit that created it, and the identification of deleted elements. Using this profile enables one to keep some traceability information in UML models. The profile can be extended to attend the needs of different transformation units and this approach does not require transformation languages to be extended, since they only need to handle standard UML profile elements.

2.3 Transformation Approaches

Since MDE supports automating both the creation and the discovery of relationships among models, model transformations can be considered as a mechanism to generate trace links. Hence, most of the transformation languages support automatic creation and usage of trace links between models, but this facility alone does not guarantee that the transformations are will be well explored to help on traceability practices.

This section discusses approaches which consider model transformations as a mechanism to collaborate with traceability in MDE.

2.3.1 Loosely Coupled Traceability (LCT)

Jouault [17] shows how traceability can be added to programs written in the ATLAS Transformation Language (ATL) [18] in order to achieve the limits of implicit traceability. ATL is a model transformation language that supports dedicated support for traceability but its trace generation mechanism is implicit. ATL has a built-in support for traceability. Such a form of traceability need not persist after executing a transformation. The author states that a single transformation program can be used in several contexts and consequently, such a program may need to be able to generate different kinds of traceability information.

The author considers the traceability information as a model, more precisely as an additional target model of a transformation program. This approach allows creating traceability elements in the same way other target model elements are created. To integrate traceability in transformation programs, transformation developers should add a target pattern element to generate an external trace link in the trace model. This, however, required manually adding the pieces of ATL code.

Since transformation programs are models, an ATL program can be transformed into another ATL program to automate this manual step. An ATL program named TraceAdder [17] automatically inserts the traceability creation code. TraceAdder operates in ATL refining mode which ATL provides as a replacement mechanism and is therefore a kind of in-place transformation. Since it is used just before actual ATL compilation, it is considered as a precompiler. One of the advantages of this solution is that traceability generating code is explicit, but not tightly coupled to program logic. From a tool support perspective, the ATL engine is a plug-in for the Eclipse and supports EMF [9] and MOF [29] models.

2.3.2 On Demand Merging of Traceability (ODMT)

Kolovos et al. [20] present an approach for merging primary models with their correspondent trace models and generate annotated models on-demand, which contain traceability information useful for inspection purposes. Generated traceability links can be stored and managed by using two different approaches. In the first approach, named embedded traceability, links are embedded inside the target models they refer to in the form of new model elements. Using this approach makes defining and understanding traceability links much easier, but it creates many model elements that do not belong to the model. In the second approach, these links are stored in separate models.

The on-demand merging of traceability links with models requires elements of the related models having a persistent identification feature and this makes the traces between model elements hard to understand. The authors suggest that traceability information should be maintained in separate models, which can be merged with the primary model(s) on demand to produce annotated models for inspection purposes. The authors also present a concrete example to automate this merging process. They produce models annotated with traceability elements on-demand to overcome the primary problem of external traceability, the lack of human-friendliness. They have two suggestions for their solutions: (a) the solution they propose must apply to all possible traceability metamodels because there is no consensus on a global traceability metamodel, (b) it must not be limited to the context of a single modelling language such as UML.

The Epsilon Merging Language (EML) [20] is used to implement the merging of models with traceability information. Model merging is completed in two phases: *matching* and *merging*. The correspondences between elements of the source models are established in the matching phase. Later, the elements identified are merged in the merging phase. From a tool support perspective, EML is a plug-in for the Eclipse and supports managing EMF [9] and MOF [29] models as well as XML documents.

2.3.3 Traceability Framework for Model Transformations (TFMT)

Falleri et al. [11] define a traceability framework for facilitating the trace of model transformations. Their framework is inspired by Jouault [17] and implemented in the model oriented language Kermeta [30]. The framework allows tracing transformation chains within Kermeta, by means of the specification and implementation (also in Kermeta) of a language independent trace metamodel. This metamodel defines a model transformation trace as a bipartite graph on which nodes are source nodes and target nodes.

In the metamodel, a transformation chain trace is represented by a trace. Every trace is an ordered set of trace steps, each of which represents a single transformation (from a source model to a target model). A step is composed of several links and these links relate target and source objects. Objects can represent every type of model element, in different granularity levels (such as classes or class properties). Composing steps under the aggregation of traces enables developers to trace model elements in a transformation chain.

The authors have implemented the following features of the traceability framework [11]: generic traceability items; trace serialization; and a simple transformation for trace visualization using Graphviz [15] (in order to visualize the resulted transformation trace chain). Despite the framework implements only a basic metamodel for transformation chain trace, the trace generating code is tangled with the transformation code on the definition of a tracing operation.

3 Evaluation of the approaches

In this section we present a comparative analysis of traceability approaches for MDE with respect to the following comparison criteria: *representation* of traceability information, *mapping* between models, *scalability*, *change impact analysis*, and *tool support*. These are general criteria that could also be used to evaluate any traceability approach in traditional software engineering. Other evaluation criteria, such as the support of rationale and alternatives at each level, are also feasible to evaluate traceability in MDE, as well as specific criteria to judge MDE particularities. Therefore, due to our aim is to identify open issues in traceability in MDE that are new or still remain with respect to traditional methods, we decided to explore in this first evaluation only some general criteria.

The *representation* criterion compares the approach's capability to represent traceability information. The *mapping* criterion analyzes whether the approach is capable of generating traces among the models at different levels of abstraction. The *scalability* criterion analyses whether the approach can be efficiently applied to large systems. The *change impact analysis* criterion evaluates whether the approach provides support for determining the impact of changes on the artefacts across the software development lifecycle. At last, the *tool support* criterion evaluates whether the approach provides any tool support for facilitating traceability.

3.1 Representation

The representation criterion observes the main structures that are used for representing traceability information by the approaches discussed in Section 2. Table 1 relates the approaches according to this criterion.

RTTC: Almeida et al. [4] represent traceability information using traceability cross-tables. These models are used to show the trace relationships associated to the application requirements. Assessment activities or conformant transformations between models are necessary to justify the check marks in positions of the cross-table.

EBT and EBT-DP: Event-based subscriptions are used to represent traceability information in EBT [5] and EBT-DP [6]. The notification of these events carries structural and semantic information concerning a change context.

As the EBT-DP [6] considers SIG models, traceability information is also represented by interdependencies among softgoals (non-functional requirements) and operationalizations (representing design patterns).

GCT: The Goal-Centric Traceability approach [7] uses softgoal interdependency graphs and makes the tracing between requirements connecting its elements (goals and operationalizations) using explicit and implicit interdependency links. A traceability matrix is also constructed to relate SIG elements with classes.

		Representation
Requirements-Driven Approaches	RTTC	Traceability cross-table (requirements X models)
	EBT	Event-based subscriptions
	GCT	SIG graph (goals, operationalizations and contribution links) and traceability matrix
	EBT-DP	SIG graph and event-based subscriptions
	RMRT	Traceability metamodels
Modelling Approaches	SDTDA	Footprint graphs
	OST	Rules, conditions and actions
	UTSS	Traceability Scheme (TS)
	PTTM	UML models
Transformation Approaches	LCT	Trace model encoded in ATL
	ODMT	EML (the metamodel) and UML (the trace model)
	TFMT	Kermeta models (the proposed metamodel) and XMI (the serialized instances of transformation trace chain)

Table 1. Representation of trace information in traceability approaches in MDE

RMRT: Ramesh and Jarke [28] use traceability reference models to represent different levels of traceability information and links. The granularity of the representation of traces depends on the expectations of the stakeholders. Their approach supports simple or more detailed traceability information representation across the low-use and high-use reference models. Implementations of these reference models present distinct ways to embody the traceability information.

SDTDA: In [10], traceability information is represented in a graph structure called a footprint graph.

OST: Aizenbud-Reshef et al. [2] outline an operational semantics of traceability relationships that capture and represent traceability information by using a set of semantic properties, composed of events, condition and actions.

UTSS: In [23], the authors analyse several traceability approaches and propose a unified Traceability Scheme (TS) specification. The TS is composed of a traceability link dataset, a traceability link type set, a minimal set of traceability links, and a metrics set for the minimal set of traceability links.

PTTM: Vanhooff and Berbers [31] have defined a UML profile that represents and supports transformation traceability links.

LCT: Trace models generated by ATL transformation programs are used in [17] to trace other models. The author considers traceability information as a model, and extends ATL programs for supporting the generation of traces during model transformations.

ODMT: Kolovos et al. [20] use external traceability links and adopt EML as a merging language for generating annotated models with traceability information. The authors use an EML trace metamodel in the merging process, which is compliant with MOF. The annotated traceability model conforms to the trace metamodel. UML class diagrams are used as an example of models that can be manipulated by their approach.

TFMT: Falleri et al. [11] represent traceability information using Kermeta models for implementing the trace metamodel of the framework and generates serialized instances of resulting transformation trace chains in XMI.

3.2 Mapping

The mapping criterion evaluates whether the approach supports traceability of model elements at different levels of abstraction. We have observed the support to *intra-level* relationships (traces among artefacts of the same abstraction level), *inter-level* relationships (traces among artefacts of different abstraction levels), or both intra and inter-level relationships. Table 2 shows the comparison of the approaches according to the mapping criterion.

RTTC: Almeida et al. [4] represent the tracing of consecutive models using a traceability cross-table. This table shows the trace relations from requirements to other models at different development phases.

EBT and EBT-DP: Both EBT [5] and EBT-DP [7] support the indirect mapping from requirements to other artefacts, using event-based mechanisms. In addition, the EBT-DP approach [7], also supports traces between softgoals when relating non-functional requirements to design patterns.

GCT: Cleland-Huang et al. [5] provide traces between softgoals (non-functional requirements) and operationalizations (functional requirements) at requirements level, using the interdependency links of a softgoal interdependency graph. Requirements and classes are related using traceability matrix.

RMRT: Ramesh and Jarke [28] focus on requirements traceability which is intended to ensure continued alignment between stakeholder requirements and system evolution. They provide reference models for representing the traceable entities and relationships. Intra-level and inter-level traceability are supported by the low and high-use metamodels, which provides mappings between requirements and many other elements (system objectives, system components, functions, etc).

SDTDA: The trace types in Egyed [10] suggest the possibility to realize both intra-level and inter-level mapping. Observable traces relate test scenarios and classes. Generated traces can relate model elements with other models, test scenarios or classes. Finally, test scenarios can also relate to model elements through hypothesized or validated traces. In addition, this approach supports both forward and reverse engineering.

OST: Aizenbud-Reshef et al. [2] propose to add operational semantics entities to the traceability metamodels to more precisely capture and represent the intended meaning of different types of traceability. The unifying traceability scheme is an interesting proposal for representing different types of traceability links in different domains and mapping the artefacts across the software development lifecycle.

UTSS: In [23], the authors propose that the minimal set of traceability links of their traceability schema must consider the links among artefacts themselves, as well as the links among a set of artefacts and the artefacts of a previous (or next) development phase.

PTTM: Vanhooff and Berbers [31] extend the semantics of UML to add traceability support to the language. A transformation traceability metamodel is mapped to UML profiles, which can be extended. Links among various UML model element can be traced.

LCT: The simple trace metamodel presented in [17] allows the establishment of trace links between any type of source and target model elements. The traces can refer to elements of the same development phase or different phases.

ODMT: In [20] traceability information is stored in separate trace models which can be merged with a correspondent primary model (from which the trace model was generated). This approach only presents a traceability method for unidirectional and inter-level traces.

TFMT: The proposed transformation trace metamodel by [11] enables the automatic tracing of model transformations. The implementation of the framework permits its application for constructing transformation chains of different sizes and in different directions. In this sense, it allows the forward, backward, intra-level and inter-level traces of transformations, depending on the definition of source and target models.

3.3 Scalability

Since real software projects become naturally larger during their development, and the software specification involves heterogeneous artefacts and presents highly complex structures, scalability is an important criterion to be considered when evaluating the usage of traceability approaches.

A scalable traceability practice is applicable for large projects as it is for small projects. We present a brief discussion on whether the approaches may be efficiently applied to large projects, under the perspective of their processes, the visualization of trace information, and their application to a larger amount of elements (broader spectrum of metamodels and models). A comparison of the approaches from the perspective of scalability is presented in Table 2.

RTTC: Almeida et al. [4] suggest that it could be possible to cluster parts of application models and generate different visualizations on a cross-table, making the approach scalable. Therefore, the absence of tool support makes the application of this approach to medium size projects difficult.

GCT: The approach presented by Cleland-Huang et al. in [5] may be capable of supporting larger projects. One possible way of performing with larger systems, is to explore the possibility of considering subgraphs of the general SIG of the system-to-be.

EBT and EBT-DP: For the Event-Based Traceability approaches [7, 6], scalability is a problem when maintaining the dynamicity of the system traceability. As the project grows, the most difficult problem is to maintain a good performance of the EBT event server.

The scalability of the EBT-DP approach [6] is questionable also, because there is no evidence that the method can be applied with success to support a more complete set of design patterns.

RMRT: The reference models (both low-use and high-use metamodels) described in [28] may be scalable due to its possible use for traceability activities in different complexity levels. Therefore, we cannot affirm whether this approach lacks scalability with respect to tool support for large-scale projects or not. The efficiency of the tools which have implemented these metamodels was not evaluated and the tools are not the focus of the approach.

SDTDA: Egyed [10] have applied the scenario-based approach for trace dependency analysis to some large-scale projects with satisfactory quality of results. The author could observe that this approach does not request a large set of input data, as well as the complexity of trace analysis is not computationally expensive.

OST: Aizenbud-Reshef et al. [2] do not present a practical application of their approach. Therefore, it may be scalable since it is associated with the UML (largely accepted and used).

UTSS: The Traceability Schema [23] is not scalable in its current form. Therefore, Limon and Garbajosa outline a strategy that may contribute to its scalability: to include in the traceability schema a set of metrics that can be applied for monitoring and verifying the correctness of traces and their management.

■ yes
□ no
▣ partially

		Mapping	Scalability	Change impact analysis	Tool support
Requirements-Driven Approaches	RTTC	inter	□	□	□
	EBT	inter	□	■	■
	GCT	intra & inter	▣	■	▣
	EBT-DP	intra & inter	□	■	■
	RMRT	intra & inter	■	■	■
Modelling Approaches	SDTDA	intra & inter	■	■	▣
	OST	inter	▣	□	□
	UTSS	intra & inter	□	□	□
	PTTM	intra & inter	▣	□	□
Transformation Approaches	LCT	intra & inter	□	□	■
	ODMT	inter	▣	□	■
	TFMT	intra & inter	□	□	■

Table 2. Mapping, scalability, change impact analysis and tool support in traceability approaches in MDE

PTTM: In [31] the UML profile mechanism is explored and traceability information can be incorporated into UML models. The scalability of this approach is limited to the scalability of UML.

LCT: Jouault [17] claims that his approach is scalable for different transformation projects in ATL. Therefore there is no evidence of the efficiency of this approach with relation to larger projects. In addition, the manipulation of traceability in model transformations in this approach requires the manual addition of code.

ODMT: Kolovos et al. [20] propose an approach which permits the merging of a model with its correspondent trace model on-demand. Traceability information is maintained in a separate model and a generic trace metamodel is used for flexibility reasons. The on-demand merging of traceability links with models may be a scalable approach due to these characteristics.

TFMT: In [11] the scalability is compromised since the authors' implementation of trace generating code is tangled with the transformation code on the definition of a tracing operation. This makes the traceability framework less reusable, the transformation rules not clear and the traceability process less transparent. There is no evidence of the efficiency of this approach with relation to larger projects (or with longer transformation trace chains). Despite of that, the metamodel may be extendable and scalable.

3.4 Change Impact Analysis

The *change impact analysis* criterion checks whether an approach determines the effect of change on the entire system and on the artefacts across the software development lifecycle. Table 2 shows the coverage of change impact analysis mechanisms by the approaches.

RTTC: Almeida et al. [4] postpone the investigation traceability of requirements in face of changes in requirements specification to a future work. The authors do not address the issues regarding the changes in requirements and their effects in the detailed design documents or source code.

EBT: The event-based traceability approach by Cleland-Huang et al. [5] supports change impact analysis. During the management of evolutionary change, a set of standard change events is defined and a method for monitoring user's actions and the recognition and publication of change events is proposed.

GCT: The GCT model [7] provides change impact analysis among functional and non-functional requirements, represented using softgoal interdependency graphs.

EBT-DP: Cleland-Huang and Schmelzer [6] present a mechanism for manipulating dynamic generated traceability links. They consider a change impact analysis and the use of regression tests. As an instance, the approach supports the identification of critical elements that should remain in the system for keeping the integrity of a traceable non-functional requirement.

RMRT: Ramesh and Jarke [28] provide means of analyzing change impact according to the description of the rationale submodel.

SDTDA: Egyed [10] presents an iterative approach to trace dependency analysis which characterizes highly interrelated relations among test scenarios, implementation classes and model elements. The interpretation of these relational dependencies is subjective, since the representation of trace links does not supply the semantic meaning of a trace. However, this approach is capable of providing the means for the analysis of change impacts.

The other evaluated approaches do not provide any mechanisms for performing change impact analysis [2, 11, 23, 17, 20, 31].

3.5 Tool Support

Tool support is fundamental for a good application of a traceability method, not only for visualization and management of manually or automatically generated traces among software artefacts, but also for the proper support for reasoning on this information. Approaches are compared from the perspective of the provisioning of any tool support in Table 2.

RTTC: Tool support is envisioned as future work by Almeida et al. [4]. They plan to consider transformation of models and conformance as traceability relationships.

EBT: The components of the event-based traceability approach [5] were implemented as client-server architecture based on the observer design pattern. Especially, the event trigger was implemented on top of the DOORS requirements management system to manually capture change events as they occurred.

GCT: The GCT model [7] has partial tool support. During change impact analysis, despite of the fact that the retrieval algorithm uses probability to return links between impacted requirements (elements in a SIG) and classes, user's appraisal is required to manage the traceability links.

EBT-DP: Cleland-Huang and Schmelzer [6] support the static and dynamic generation of traceability links across the development phases, although only a few characteristics of their approach are fully implemented.

RMRT: The reference metamodels for traceability by Ramesh and Jarke [28] were encoded in a knowledge-based meta database management system called ConceptBase. Later, they were adopted in several commercial traceability tools, such as SLATE and Tracenet.

SDTDA: In the scenario-driven approach by Egyed [10], the activities for scenario-testing and finding hypothesized traces are manual, while trace analysis and result interpretation are automated.

OST: In [2] the authors present an approach for considering operational semantics of traceability in UML and claim that their work can be applied to MOF, but no implementation or tool support is given.

UTSS: Limon and Garbajosa [23] propose a traceability schema, but there is no tool support yet for the employment of their approach.

PTTM: The approach presented in [31] represents a traceability link using stereotype specifications and is independent on model transformation languages. The authors do not relate the existence of a tool that gives support to their approach.

LCT: Tool support and scalability are the key criteria for transformation approaches. Jouault's traceability study was implemented in ATL [17, 18] and this language is supported by Eclipse.

ODMT: In Kolovos et al. [20], EML [21] is used to implement the merging of models with traceability links. EML specifications can be managed by plug-ins for Eclipse.

TFMT: Falleri et al. [11] have implemented the transformation chain trace metamodel in a model-oriented language compatible with EMF [9] called Kermeta [30], and some graphical visualization of the trace in Graphviz [15].

4 Open issues

From the comparative analysis of the approaches for model traceability we identify the following open issues:

- In the early development stages in MDE, less automation is found to cope with traceability. The degree of automation of traceability practices, mainly in the early stages of software development in MDE, is an open issue due to the fact that there is the need of more appropriate models (for requirements, features, goals, etc) in early stages. Depending on the way MDE is applied, as the structure of models is well defined, the capture of trace information in trace models may be facilitated and automated due to the knowledge about models which conform to metamodels.

- Building better trace metamodels for enabling traceability and make use of the facilities provided by model-driven techniques should be better explored by traceability approaches in MDE. The semantics of traceability models and their structure is an open issue. Although this aspect is absolutely independent of MDE processes or other development processes, MDE may help on the automatic creation of traceability links on the basis of a metamodel that presents a good taxonomy of trace dependencies and that expresses transitivity of trace dependencies.

- In traditional approaches, the variety of models used for describing different artefacts (which usually do not conform to any metamodel) makes it difficult to manage fine grained trace links due to the heterogeneity of the models. Moreover, since different tools are used during the various traceability steps throughout the software development stages, traditional traceability processes have more troubles to collect and manage trace information, as well as to keep it consistent. In MDE, if the metamodels can be described using compatible modelling languages one can create more uniform manners to perform automated traceability management during the development stages. Work should be done in this direction.

- The issue on enabling the connection between transformation programs and traceability mechanisms and provide the automated update of target models in the case of changes in source models deserves more attention. The tool support offered by the approaches was somehow incomplete, as the models are not traced throughout the complete software life-cycle. Transformation engines and transformation languages should address this issue more explicitly.

- It is still unknown whether trace models and incremental model transformation [16] can support each other efficiently. For instance, trace models should consider information (e.g. constraints) about a certain transformation. If we do not want to execute the complete transformation again, how do we cope with the update of the target model using a less costly procedure?

- Another open issue is the discovery of traces between model elements when these traces are not described explicitly. How to deal with the implicit relationships between models? What kinds of trace information a model brings from its own sources to its targets? The evaluated traceability approaches do not explore mechanisms for discovering important implicit traces.

- Mechanisms for the evolution of trace links are not explored by the majority of the observed approaches. Our viewpoint is that the automatic update of trace models by using model transformations (or not) should be considered as a way to keep the consistency of the traceability information and then achieve a high quality traceability process. Traces should evolve as models and transformations evolve.

5 Conclusions

In this paper we have built on the evaluation of traceability approaches for identifying the open issues on tracing requirements and model elements in MDE. We have discussed the state-of-the-art in traceability approaches in MDE and evaluated them according to five general comparison criteria for traceability: *representation*, *mapping*, *scalability*, *change impact analysis* and *tool support*. Based on our observations, we have presented a set of open issues that propose a better exploration of MDE characteristics and some ways on which it can cope better with traceability.

This survey shows that tool support is crucial to automate the traceability in MDE. The automation of traceability mechanisms can be simplified in MDE by considering conformant transformation specifications as a way to provide traceability information [1]. In addition, the representation of trace information plays a crucial role in achieving the benefits of applying traceability techniques. The taxonomy of trace links is independent on the MDE paradigm, therefore the model-driven techniques can help on the specification of good trace metamodels that will cover specific needs. Some factors that inhibit the automation of traceability practices in MDE are the absence, impreciseness or inconsistence of information concerning model elements and trace links.

In addition, traceability support may not be a property of a transformation language. It may be provided by the transformation engine or the developer may take care of creating and using traces. A promising direction is the use of hybrid approaches (with static and dynamic generated traceability links) [8] and metamodels for external traceability links [20]. Moreover, the dynamic trace generation is fundamental in MDE for supporting chains of model transformations [11].

We also observed that the representation of external traceability links, stored in separate models that can be combined with the primary models they refer to, facilitates the loose coupling between models and traceability information. As a consequence, a more flexible traceability mechanism can be created to allow inspection and decision making during MDE processes.

6 Acknowledgements

The research presented in this paper has been carried out in conjunction with the AMPLE Project IST-33710 (http://www.ample-project.net/) and QuadREAD project (http://quadread.ewi.utwente.nl/). This work has profited from the input we received by Pim van den Broek, Klaas van den Berg and Ivan Kurtev on an earlier discussion on this survey.

References

[1] N. Aizenbud-Reshef, B. T. Nolan, J. Rubin, and Y. Shaham-Gafni. Model Traceability. *IBM Systems Journal*, 45(3):515–526, 2006.

[2] N. Aizenbud-Reshef, R. F. Paige, J. Rubin, Y. Shaham-Gafni, , and D. S. Kolovos. Operational Semantics for Traceability. In *ECMDA-TW: Traceability Workshop, at European Conference on Model Driven Architecture*, pages 7–14, Nuremberg, Germany, November 2005.

[3] M. Alanen, J. Lilius, I. Porres, and D. Truscan. Model driven engineering: A position paper. In J. M. Fernandes, J. Lilius, R. J. Machado, and I. Porres, editors, *Proceedings of the 1st International Workshop on Model-Based Methodologies for Pervasive and Embedded Software MOMPES'04*, number 29 in General Publications, pages 25–29. Turku Centre for Computer Science, Jun 2004.

[4] J. P. Almeida, P. van Eck, and M.-E. Iacob. Requirements Traceability and Transformation Conformance in Model-Driven Development. *Tenth IEEE International Enterprise Distributed Object Computing Conference (EDOC'06)*, pages 355–366, 2006.

[5] J. Cleland-Huang, C. K. Chang, and M. Christensen. Event-Based Traceability for Managing Evolutionary Change. *IEEE Transactions on Software Engineering*, 29(9):796–810, 2003.

[6] J. Cleland-Huang and D. Schmelzer. Dynamically Tracing Non-Functional Requirements through Design Pattern Invariants. In *Proceedings of the Second International Workshop on Traceability in Emerging Forms of Software Engineering (TEFSE'03)*, 2003.

[7] J. Cleland-Huang, R. Settimi, O. BenKhadra, E. Berezhanskaya, and S. Christina. Goal-centric Traceability for Managing Non-functional Requirements. In *Proceedings of the 27th International Conference on Software Engineering (ICSE'05)*, pages 362–371, 2005.

[8] K. Czarnecki and S. Helsen. Classification of Model Transformation Approaches. In J. Bettin, G. van Emde Boas, A. Agrawal, E. Willink, and J. Bezivin, editors, *Proceedings of the Second Workshop on Generative Techniques in the Context of Model-driven Architecture (OOPSLA'03)*. ACM Press, October 2003.

[9] Eclipse.org. Eclipse Modelling Framework. Available at http://www.eclipse.org/emf.

[10] A. Egyed. A Scenario-Driven Approach to Trace Dependency Analysis. *IEEE Transactions on Software Engineering*, 29(2):116–132, 2003.

[11] J. Falleri, M. Huchard, and C. Nebut. Towards a Traceability Framework for Model Transformations in Kermeta. In *ECMDA-TW: Traceability Workshop, at European Conference on Model Driven Architecture*, pages 31–40, Bilbao, Spain, July 2006.

[12] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, January 1995.

[13] A. Geraci. *IEEE Standard Computer Dictionary: Compilation of IEEE Standard Computer Glossaries*. Institute of Electrical and Electronics Engineers Inc., The, 1991.

[14] O. C. Z. Gotel and A. C. W. Finkelstein. An Analysis of the Requirements Traceability Problem. In *Proceedings of the International Conference on Requirements Engineering*. IEEE Computer Science Press, 1994.

[15] Graphviz. Graph Visualization Software. Available at: http://www.graphviz.org.

[16] D. Hearnden, M. Lawley, and K. Raymond. Incremental model transformation for the evolution of model-driven systems. In *MoDELS*, pages 321–335, 2006.

[17] F. Jouault. Loosely Coupled Traceability for ATL. In *ECMDA-TW: Traceability Workshop, at European Conference on Model Driven Architecture*, pages 29–37, Nuremberg, Germany, November 2005.

[18] F. Jouault and I. Kurtev. Transforming Models with ATL. In *Proceedings of the Model Transformations in Practice Workshop, at MoDELS'05*, Montego Bay, Jamaica, 2005.

[19] S. Kent. Model driven engineering. In *IFM '02: Proceedings of the Third International Conference on Integrated Formal Methods*, pages 286–298, London, UK, 2002. Springer-Verlag.

[20] D. Kolovos, R. Paige, and F. Polack. Merging Models with the Epsilon Merging Language. In *Proceedings of ACM/IEEE 9th International Conference on Model Driven Engineering Languages and Systems (Models/UML'06)*, LNCS, Genoa, Italy, October 2006. Springer Verlag.

[21] D. S. Kolovos, R. F. Paige, and F. Polack. On-Demand Merging of Traceability Links with Models. In *ECMDA-TW: Traceability Workshop, at European Conference on Model Driven Architecture*, Bilbao, Spain, July 2006.

[22] I. Kurtev. *Adaptability of Model Transformations*. Phd thesis, IPA, 2005. ISBN 90-365-2184-X.

[23] A. E. Limon and J. Garbajosa. The Need for a Unifying Traceability Scheme. In *ECMDA-TW: Traceability Workshop, at European Conference on Model Driven Architecture*, pages 47–55, Nuremberg, Germany, November 2005.

[24] L. Naslavsky, T. A. Alspaugh, D. J. Richardson, and H. Ziv. Using Scenarios to Support Traceability. In *TEFSE '05: Proceedings of the 3rd International Workshop on Traceability in Emerging Forms of Software Engineering*, pages 25–30, New York, NY, USA, 2005. ACM Press.

[25] Object Management Group. UML 2.0 Superstructure Specification, 2004. Available at: http://www.omg.org/cgi-bin/doc?ptc/2004-10-02.

[26] B. Paech and A. von Knethen. A Survey on Tracing Approaches in Practice and Research. Technical Report IESE-Report Nr. 095.01/E, Fraunhofer - Institut Experimentelles Software Engineering, 2002.

[27] J. D. Palmer. Traceability. *Software Requirements Engineering*, pages 364–374, 1997.

[28] B. Ramesh and M. Jarke. Toward Reference Models for Requirements Traceability. *IEEE Transactions on Software Engineering*, 27(1):58–93, 2001.

[29] Sun Microsystems. Meta Data Repository. Available at: http://mdr.netbeans.org.

[30] Triskel project (IRISA). The Metamodeling Language Kermeta. Available at: http://www.kermeta.org.

[31] B. Vanhooff and Y. Berbers. Supporting Modular Transformation Units with Precise Transformation Traceability Metadata. In *ECMDA-TW: Traceability Workshop, at European Conference on Model Driven Architecture*, Nuremberg, Germany, November 2005.