Towards a Multi-Domain Model-Driven Traceability Approach

Masoumeh Taromirad, Nicholas Matragkas, and Richard F. Paige

Department of Computer Science, University of York, UK
[mt705,nicholas.matragkas,richard.paige]@york.ac.uk

Abstract. Traceability is an important concern in projects that span different engineering domains. In such projects, traceability can be used across the engineering lifecycle and therefore is *multi-domain*, involving heterogeneous models. We introduce the concept and challenges of *multi-domain traceability* and explain how it can be used to support traceability scenarios. We describe how to build a multi-domain traceability framework using Model-Driven Engineering. The approach is illustrated in the context of the safety-critical systems engineering domain where multi-domain traceability is required to underpin certification arguments.

1 Introduction

Traceability is a key element of any rigorous software development process, providing critical support for many development activities. In some cases, traceability is mandated so as to comply with regulations, e.g., in civil aviation projects. However, there are substantial challenges associated with its use in practice, including identifying the most appropriate artefacts to trace. This makes it difficult to define a generic and effective *traceability framework* – consisting of a Traceability Information Model (TIM), traceability information, and analysis tools – to be used to manage trace information for a specific project; such frameworks are thus still rarely defined and used [1].

In many contexts, such as projects developing high-assurance software systems, different kinds of traceability are mandated [2]. Such projects address multiple engineering domains (e.g., software, mechanics and safety). Each domain has its own stakeholders, artefacts, tools, and goals. Stakeholders of any single domain may be concerned with both intra- and inter-domain traceability. For example, a software developer will be interested in traces from system to software requirements, while a safety engineer will want to trace relationships between fault tree analysis, software requirements and verification artefacts. Considering that traceability may be required throughout the project lifecycle, any traceability framework needs to operate across the project's different domains. In this respect, traceability is a *multi-domain* concern.

The core element of any traceability framework is a traceability information model (TIM) which provides guidance as to which artefacts to collect and which relations to establish in order to support traceability goals [3]. Traceability

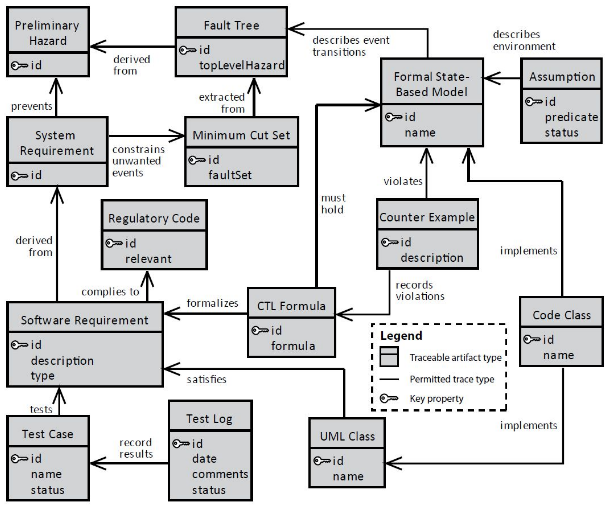

Fig. 1. A traceability information model for a safety-critical system [2]

goals, e.g., 'traceability of designs against requirements' and 'track the allocation of requirements to system components', specify purposes for accumulating traceability data. A TIM may refer to artefacts (documents, models, databases, project activities context) from different domains or require relationships between multiple domains.

The main contribution of this research is a model-driven approach to support multi-domain traceability. It introduces detailed steps that can be used to build a multi-domain traceability framework. We express a TIM as a domain-specific modelling language and use model-driven engineering (MDE) techniques to derive traceability information from sources, record the information in a traceability model (TM), and finally perform traceability analyses, based on traceability goals. Section 3 describes the steps in more detail and gives potential ways in which MDE can help support them.

2 Motivation

To introduce our approach, we give an example from the safety-critical systems domain. We then highlight the open challenges for multi-domain traceability.

2.1 Example

Figure 1 depicts a basic TIM for a safety-critical system using a UML class diagram [2]. The traceable entities include artefacts from two domains: software development and safety engineering.

Typical software development artefacts are seen along the left side of the diagram: for example, software requirements are derived from system requirements; classes are designed according software requirements and implemented by code. Meanwhile, safety engineering requires additional artefacts to be produced and traced to the general software development artefacts; these are shown mainly on the right hand side of the diagram. For example, the preliminary hazard artefact documents hazards that could lead to system failure. Such hazards are modelled in more detail in a fault tree which looks at events that could lead to the hazards. System Requirements are specified to prevent hazards from occurring by preventing the unwanted events documented in the Minimum Cut Sets. The Software Requirements may also have to comply with Regulatory Codes.

Although the TIM captures all the traceable components in one metamodel and ultimately will be instantiated in a single traceability model (TM), most of the traceability information needed to build the TM is available in different domains. For example, the relationship between 'System Requirement' and 'Software Requirement' is an elementary trace link specified in general software development projects (regardless of the type of the project). The link between 'Formal State-Based Model' and 'Assumption' is a link type normally provided in the safety engineering domain. Accordingly, each domain includes traceability information related to that domain.

In this respect, to capture traceability information and generate a traceability model (TM), we need to find ways that (re-)use existing information and minimises rework.

2.2 Challenges

A review of the literature suggests several challenges for multi-domain traceability not fully addressed by existing approaches.

Domain-specific Traceability Information Traceability information is captured and collected based on a TIM. As illustrated in the above example, each domain includes traceability information specific to that domain. For a systems engineering project, each domain (and hence each TIM) can provide part of the information needed to generate a complete traceability model. In this respect, local traceability information is essential to capture and record project-wide traceability information, though there is no guarantee that it will be sufficient to achieve traceability goals.

Heterogeneous Traceability Information Usually, available traceability information is provided in different formats, including documents (plain text or structured languages), models (e.g. UML class diagrams), databases, tools, or XML documents. To collect the traceability information using existing available information, we need to find a systematic approach to extract the required information and integrate them as the ultimate traceability information. In this context, integration of information from various domains which are expressed in different and heterogeneous formats is an important concern.

Missing Inter-domain Traceability Information As mentioned earlier, we cannot just rely on the available information as it normally does not cover inter-domain information. Usually, the relationships between domains are defined informally or incompletely which results in inconsistencies and redundancies among domains. Specifying and recording the inter-domain relations are of the essential needs to accumulate the traceability information.

Separation of Concern (SoC) People are interested in their own domain and usually prefer to work with familiar tools or techniques; the existing tools, models, and techniques for a specific domain which are specialised for that domain. Therefore, it is not reasonable to require all stakeholders to work with the traceability model directly.

Tool Support Tool support for a traceability framework is essential to maximise the return on investment in building a TIM. However, practical guidance on how to define and implement a TIM, and use it in practice, is still a poorly understood issue [1]; the effort needed for specifying and managing a TIM and the tools which let the user implement it are the main concerns of supporting traceability.

3 Multi-Domain Traceability

This section introduces our approach to support multi-domain traceability. It defines a traceability framework, and discusses how a model-driven approach to traceability can help to effectively support the approach. Our approach constructs a modelling language to describe the traceability information model (TIM). We identify and specify the relationships between TIM and existing project information sources. Traceability information is captured and instantiated in a *single* traceability model (TM) that conforms to the TIM and is used to perform traceability analyses. Fig. 2 illustrates the proposed approach.

3.1 Traceability Information Model

The core element of any traceability framework is a TIM, which identifies the information required to support traceability goals, such as which artefacts should be traced, the level of detail of the traces, and how traceability links should be classified regarding their usage, context or semantics [3]. So, the first step to define a traceability framework is to define a suitable TIM that supports project traceability goals.

In our approach, the TIM is described as a modelling language, and is thereafter used to generate traceability models. Fig. 1 shows an example TIM described as a UML class diagram.

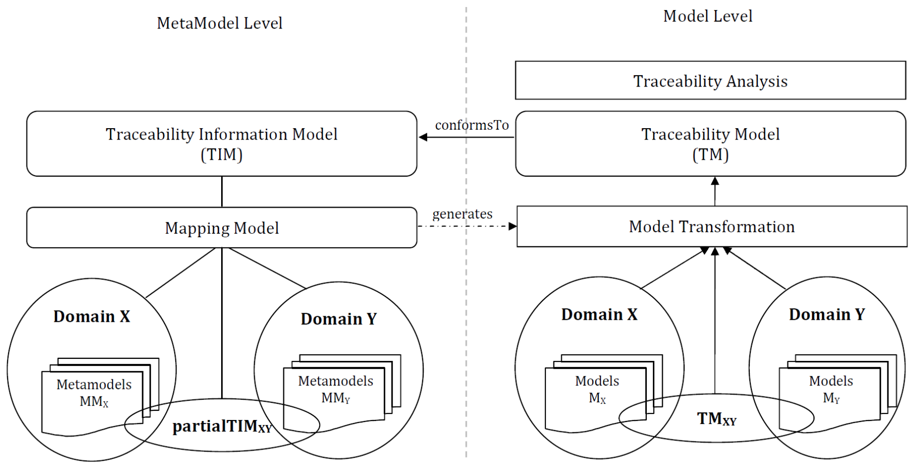

Fig. 2. The proposed model-driven approach to multi-domain traceability

3.2 Traceability Information

Once the TIM has been defined, traceability information is collected and recorded in a traceability model. In our approach, the traceability model is built on top of the other models, generated automatically (by a query), not containing any information that cannot be regenerated automatically. We consider the TM as a *view* similar to 'view' in database context in which view is defined as a dependent object over some tables and theoretically generated on-demand. A single TM provides a coherent view of the traceability information; using a diverse set of traceability information sources (usually represented in heterogeneous formats) is one of the main problems in working with trace links [4]. The TM unifies the way in which the traceability information can be used to perform traceability analyses.

We propose the following steps and activities to build the traceability model:

Step 1: Identify the available information. Based on the TIM, available information sources are gathered to find out how much of the required information is provided and available, in which ways, and how it can be used. As a result, the available information (models, trace link types both within and between domains) and missing information (models, trace link types, . . .) is identified.

Step 2: Add the missing information. Based on the results of step 1, we complete the information sources to provide the missing information. The missing information can be divided into two parts: information limited to one domain and that which relates to multiple domains, such as inter-domain trace link types. We elaborate on this in more detail.

Step 2.1: Add the missing information from one domain. To complete the missing information in one domain, the following options are available:

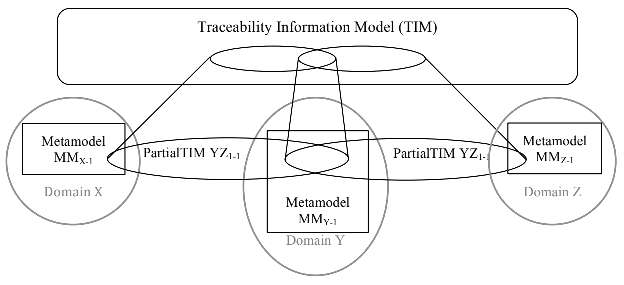

Fig. 3. Partial Traceability Information Models

- complete/extend existing metamodels in each domain by adding missing objects, trace link types, and validation rules, and then update or regenerate the models.
- define new metamodels and create new models within one domain, whenever required.

Step 2.2: Add the inter-domain missing information. After completing the required information for each domain, the inter-domain information are considered. One of the main missing information would be the trace link types defined between domains which can not be easily captured and recorded. The available trace link types are usually limited to one domain. To represent the inter-domain traces, we propose building a traceability model between pairs of domains, wherever required, to provide the missing trace links. To do this, we need to define a traceability information model for each required traceability model. Each traceability information model – a *partial* TIM – is a submodel of the main TIM. Fig. 3 shows the relationship between partial TIMs and the main TIM.

Step 3: Define the mapping between TIM and the information sources. Once all the source models for the traceability information are available, we define the *mapping* between these models and the TIM. The mapping explicitly specifies how each concept (object and trace link type) in the TIM is related to concepts in the source models. The mapping is used to collect information and generate the traceability model. The mapping is also used to interpret the result of traceability analyses, performed on TM, in terms of source models in different domains. The *mapping* is similar to a Correspondence Model (CM) in model composition [5]. A CM is a model that explicitly describes the relationships between elements of different models, but is constructed specifically for model comparison or merging processes.

Step 4: Generate the traceability information. Finally, based on the mapping, the traceability model is generated (as depicted in Fig. 4) The traceability model is created so as to minimise redundancy and inconsistency; it captures the min-

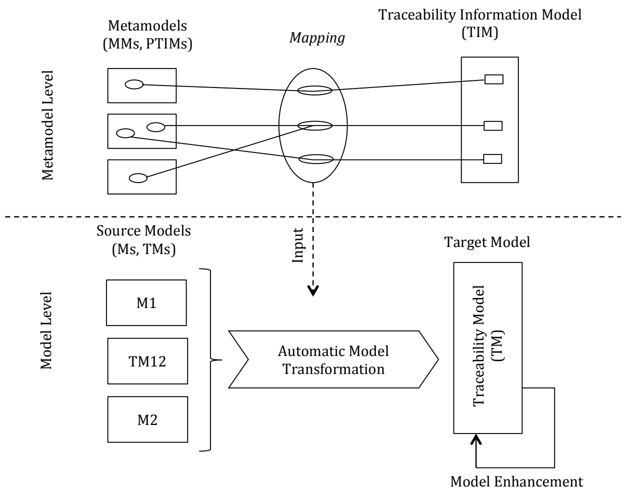

Fig. 4. Generating the traceability model (TM)

imum information needed. For example, it may just contains reference to the source elements in the source models instead of redefining these elements.

As mentioned before, the required traceability information could be represented in various formats (e.g. plain text, XML files) with different underlying structures and metamodels. To support analyses and an overall MDE approach to developing tools, we need to provide a MDE view of the non-model information (wherever it is possible and reasonable) as a prerequisite of building the TM; this can be challenging. However, we focus on those models that are available and that information which can be automatically transformed to models: there are several types of model in different domains (e.g., requirements models, safety analysis models) that provide substantial traceability information. Our approach does not limit the types of model that can be considered in the traceability framework.

3.3 Traceability Analysis

Traceability information is captured and recorded to support traceability goals. Usually, traceability goals are explained in very abstract terms. For example, they can be expressed 'traceability of designs against requirements' or 'track the allocation of requirements to system components'. To be able to support the goals, we need to define concrete traceability analyses which can be applied on the traceability information to determine whether traceability goals are satisfied. In this way, each traceability goal may result in one or more traceability analyses which are defined in terms of traceability and the TIM.

We can use generic query languages (e.g. SQL) and model management languages to express the traceability analyses, which require knowledge of the underlying structures in which the traceability information is stored. As an alternative, we suggest defining a task-specific query language to express traceability analyses precisely. A task-specific query language – bound to the TIM – would hide the underlying complexity and diversity of the underlying information representation.

3.4 Implementation

Typically, an implementation of a TIM will be in the form of a *traceability metamodel*, which will be used to create traceability models. It is these TMs that model the trace links between concepts and artefacts in a project.

In our approach, the TIM is defined as a modelling language; this can be done with any metamodelling technology. Model management tools can then be used to query and manipulate traceability models. For example, the Ecore metamodelling language [6] and Epsilon [7] can be used to describe the traceability metamodel and work with models, respectively. We implemented a basic TIM (for a safety case study) using Ecore, and used Epsilon (specifically EOL, ETL, EVL) to build a TM, query the traceability model, execute constraints on models, and generate analysis reports.

3.5 Discussion

The model-driven approach to TIM definition and implementation enables us to work with arbitrary engineering models and effectively use them. The approach uses the domain-specific traceability information, extract the required information from them regarding the TIM, and generate a TM for the project. The TM is a *view* built automatically through model management operations over the available models in different domains. Throughout the process to generate the TM, missing traceability information, mainly inter-domain traces, is identified and added to the existing information.

The TIM and TM will allow the traceability users to ignore the underlying information complexity, data structures, and information representation format of artefacts, instead allowing them to focus on achieving traceability goals. The proposed approach also supports separation of concerns: different artefacts from different domains that are being traced do not need to be combined (possibly artificially) in one overall description, and can be managed separately while traceability information is defined.

One of the main concerns with traceability implementations and tools is managing *change*, for example, changes in the TIM. The traceability framework that we have developed is also subject to change and evolution. Based on an analysis of the artefacts involved in the traceability framework, and the types of change we can encounter in MDE, we focus on the following changes to the traceability framework: change in TIM, change in domain metamodels, change in domain models. Change in the TIM is the most expensive change as it requires updating most of the involved artefacts (e.g. models, metamodels, and mapping). Change in the metamodels in each domain results in change in the intermediate models and the mapping. Finally for the change in domain models, the traceability model is regenerated automatically based on the mapping and new models.

4 Related Work

There are challenges associated with defining a TIM, such as finding the appropriate level of granularity; as such, TIMs are often considered to be project specific. Researchers agree on the value of a project-level definition of a TIM as it facilitates a consistent and ready-to-analyse set of traceability relations for a project [4]. As discussed in [8], project characteristics are critical in finding the *necessary and sufficient* amount of required information which should be recorded to support the traceability goals.

[9] highlights the importance of a project-specific TIM and suggests a UML-based approach to define, implement, and use it. [2] proposes a usage-centred traceability process which uses UML class diagrams to define traceability strategies for a project. [2, 10–12] each focus on traceability in the safety domain and propose traceability metamodels and queries for that domain.

Another strand of traceability research is on reducing the effort associated with managing traceability. Egyed et al. [8] introduce value-based requirements traceability to balance cost and benefits related to capturing and maintaining traceability. [13] proposes dynamic requirements traceability to minimize the need for creating and maintaining explicit links and reduce the effort required to perform manual trace. In this context, some studies take different approaches, such as improving tools in order to decrease the cost of providing traceability. [14] provides a tool-based approach for agile requirements capture and traceability.

5 Conclusion and Future Work

This paper introduced *multi-domain* traceability and highlighted its fundamental concerns. Traceability is multi-domain as it is often required to capture the artefacts and trace links either within one or across many domains. We presented a model-driven approach to constructing and managing a framework to support traceability activities. We showed how to define a TIM and express it as a traceability metamodel, to be used later to capture and record traceability information.

We observed that there are several available information sources (mainly models) which provide a considerable amount of required traceability information. A traceability model is generated as a *view* over all the other project models, providing a coherent view of traceability, and unifying the way in which information is used (e.g. traceability analyses). The *mapping* between the TM and the other models is specified and used to generate the traceability model.

We identified that change is an important concern and we need to provide more detailed and precise support. Existing model migration and co-evolution techniques could be helpful to cope with change effectively. It may also be fruitful to create TIMs using the traceability metamodelling language approach in [15].

Improved tool support for TIMs and TMs is needed. Currently, the TIM can be implemented as a metamodel in an arbitrary technology, and then existing model management languages can be used. But, as discussed in Section 3.4, providing traceability-specific tools and technical support would improve the traceability framework. A *Traceability Query Language* (TQL) to describe the traceability analyses, a DSL to specify the mapping between the main TIM and the other metamodels, and automatic generation of transformation rules based on the mapping are examples of potential improved support. We plan to work on this next while rolling out detailed case studies in domains that require traceability (particularly safety, health informatics, and security).

References

1. Mäder, P., Gotel, O., Philippow, I.: Motivation Matters in the Traceability Trenches. In: Proc. RE'09. (2009) 143–148
2. Cleland-Huang, J., Heimdahl, M., Hayes, J.H., Lutz, R., Maeder, P.: Trace Queries for Safety Requirements in High Assurance Systems. In: Proc. REFSQ'12. (2012) 179–193
3. Ramesh, B., Jarke, M.: Toward Reference Models for Requirements Traceability. IEEE Transactions on Software Engineering **27** (2001) 58–93
4. Mäder, P., Cleland-Huang, J.: A Visual Traceability Modeling Language. In: Proc. MoDELS'10. (2010) 226–240
5. Bézivin, J., Bouzitouna, S., Fabro, M.D.D., Gervais, M.P., Jouault, F., Kolovos, D., Kurtev, I., Paige, R.F.: A canonical scheme for model composition. In: Proc. ECMDA-FA'06. (2006) 346–360
6. Steinberg, D., Budinsky, F., Paternostro, M., Merks, E.: EMF: Eclipse Modeling Framework. 2. edn. Addison-Wesley, Boston, MA (2009)
7. Kolovos, D., Rose, L., Paige, R.: The Epsilon Book. (2010)
8. Egyed, A., Grunbacher, P., Heindl, M., Biffl, S.: Value-Based Requirements Traceability: Lessons Learned. In: Proc. RE'07. (2007) 240–257
9. Mäder, P., Gotel, O., Philippow, I.: Getting Back to Basics: Promoting the Use of a Traceability Information Model in Practice. In: Proc. TEFSE'09, IEEE Computer Society (2009) 21–25
10. Peraldi-Frati, M.A., Albinet, A.: Requirement Traceability in Safety Critical Systems. In: Proc. CARS'10, ACM (2010) 11–14
11. Sanchez, P., Alonso, D., Rosique, F., Alvarez, B., Pastor, J.: Introducing Safety Requirements Traceability Support in Model-Driven Development of Robotic Applications. IEEE Transactions on Computers **60**(8) (2011) 1059 –1071
12. Katta, V., Stlhane, T.: A Conceptual Model of Traceability for Safety Systems. Technical report, Laboratory of Algorithmics, Complexity and Logic (2012)
13. Cleland-Huang, J., Settimi, R., Duan, C., Zou, X.: Utilizing Supporting Evidence to Improve Dynamic Requirements Traceability. In: Proc. RE'05, IEEE Computer Society (2005) 135–144
14. Lee, C., Guadagno, L.: FLUID: Echo Agile Requirements Authoring and Traceability. In: Proc. MWSEC'03. (2003) 50–61
15. Matragkas, N.: Establishing and Maintaining Semantically Rich Traceability: A Metamodelling Approach. PhD thesis, University of York (2011)